ETSI TS 129 501 V18.7.0 (2025-01)

TECHNICAL SPECIFICATION

**5G;
5G System;
Principles and Guidelines for Services Definition;
Stage 3
(3GPP TS 29.501 version 18.7.0 Release 18)**

Reference

RTS/TSGC-0429501vi70

Keywords

5G

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

Important notice

The present document can be downloaded from the
ETSI Search & Browse Standards application.

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 on ETSI deliver repository.

Users should be aware that the present document may be revised or have its status changed,
this information is available in the Milestones listing.

If you find errors in the present document, please send your comments to
the relevant service listed under Committee Support Staff.

If you find a security vulnerability in the present document, please report it through our
Coordinated Vulnerability Disclosure (CVD) program.

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

ETSI

Intellectual Property Rights

Essential patents

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

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

Trademarks

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

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™**, **LTE™** and **5G™** logo are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Legal Notice

This Technical Specification (TS) has been produced by ETSI 3rd Generation Partnership Project (3GPP).

The present document may refer to technical specifications or reports using their 3GPP identities. These shall be interpreted as being references to the corresponding ETSI deliverables.

The cross reference between 3GPP and ETSI identities can be found at 3GPP to ETSI numbering cross-referencing.

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Contents

Foreword

This Technical Specification has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

x   the first digit:

    1   presented to TSG for information;

    2   presented to TSG for approval;

    3   or greater indicates TSG approved document under change control.

y   the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.

z   the third digit is incremented when editorial only changes have been incorporated in the document.

In the present document, modal verbs have the following meanings:

shall   indicates a mandatory requirement to do something

shall not   indicates an interdiction (prohibition) to do something

The constructions "shall" and "shall not" are confined to the context of normative provisions, and do not appear in Technical Reports.

The constructions "must" and "must not" are not used as substitutes for "shall" and "shall not". Their use is avoided insofar as possible, and they are not used in a normative context except in a direct citation from an external, referenced, non-3GPP document, or so as to maintain continuity of style when extending or modifying the provisions of such a referenced document.

should   indicates a recommendation to do something

should not   indicates a recommendation not to do something

may   indicates permission to do something

need not   indicates permission not to do something

The construction "may not" is ambiguous and is not used in normative elements. The unambiguous constructions "might not" or "shall not" are used instead, depending upon the meaning intended.

can   indicates that something is possible

cannot   indicates that something is impossible

The constructions "can" and "cannot" are not substitutes for "may" and "need not".

will   indicates that something is certain or expected to happen as a result of action taken by an agency the behaviour of which is outside the scope of the present document

will not   indicates that something is certain or expected not to happen as a result of action taken by an agency the behaviour of which is outside the scope of the present document

might   indicates a likelihood that something will happen as a result of action taken by some agency the behaviour of which is outside the scope of the present document

might not indicates a likelihood that something will not happen as a result of action taken by some agency the behaviour of which is outside the scope of the present document

In addition:

is (or any other verb in the indicative mood) indicates a statement of fact

is not (or any other negative verb in the indicative mood) indicates a statement of fact

The constructions "is" and "is not" do not indicate requirements.

1   Scope

The present document defines design principles and documentation guidelines for 5GC SBI APIs. These principles and guidelines should be followed when drafting the 5G System SBI Stage 3 specifications.

2   References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.

- For a specific reference, subsequent revisions do not apply.

- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

[1]   3GPP TR 21.905: "Vocabulary for 3GPP Specifications".

[2]   3GPP TS 29.500: "5G System; Technical Realization of Service Based Architecture; Stage 3".

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

[4]   OpenAPI: "OpenAPI Specification Version 3.0.0", https://spec.openapis.org/oas/v3.0.0.

[5]   3GPP TS 29.571: "5G System; Common Data Types for Service Based Interfaces Stage 3".

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

[7]   IETF RFC 7396: "JSON Merge Patch".

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

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

[10]   IETF RFC 5789: "PATCH Method for HTTP".

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

[12]   IANA: "HTTP Status Code Registry at IANA", http://www.iana.org/assignments/http-status-codes.

[13]   IETF RFC 9113: "HTTP/2".

[14]   Fielding, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000.

[15]   Erik Wilde, Cesare Pautasso, REST: From Research to Practice, Springer.

[16]   YAML 1.2: "YAML Ain't Markup Language", http://yaml.org.

[17]   Semantic Versioning Specification: https://semver.org.

[18]   3GPP TS 29.510: "5G System; Network Function Repository Services; Stage 3".

[19]   IETF RFC 9457: "Problem Details for HTTP APIs".

[20]   3GPP TS 29.502: "5G System; Session Management Services; Stage 3".

[21]   3GPP TS 29.509: "5G System; Authentication Server Services; Stage 3".

[22]   3GPP TS 33.501: "Security architecture and procedures for 5G system".

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

[24]   3GPP TS 29.573: "5G System; Public Land Mobile Network (PLMN) Interconnection;Stage 3".

[25]   3GPP TR 21.900: "Technical Specification Group working methods".

[26]   IETF RFC 5234: "Augmented BNF for Syntax Specifications: ABNF".

[27]   3GPP TS 23.003: "Numbering, addressing and identification".

[28]   3GPP TS 29.503: "5G System; Unified Data Management Services; Stage 3".

[29]   IETF RFC 2046: "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types".

3   Definitions and abbreviations

3.1   Definitions

For the purposes of the present document, the terms and definitions given in 3GPP TR 21.905 [1] and the following apply. A term defined in the present document takes precedence over the definition of the same term, if any, in 3GPP TR 21.905 [1].

Absolute URI: Absolute URI follows generic URI syntax and consists of a hierarchical sequence of the following components: the "scheme", "authority", "path" and "query", i.e. excluding the "fragment" component. See clause 4.3 in IETF RFC 3986 [9].

apiRoot: apiRoot follows an absolute URI syntax, but excludes the following absolute URI identifiers: the "query" and "fragment" components. The API root contains the "scheme" and the "authority" components and may also contain an API prefix subcomponent. See clause 4.4.1. In 3GPP APIs, the default apiRoot value is "https://example.com" (see e.g. Annex A.2 in 3GPP TS 29.502 [20]), where "example.com" is replaced by the operator specific FQDN (for FQDN examples see clause 28.3.2 in 3GPP TS 23.003 [27]).

API Prefix: API prefix is an optional, deployment-specific string, which is a subcomponent of the "apiRoot". API prefix format follows "path-absolute" syntax and that starts with a "/" reserved character (see clause 4.4.1and also clause 3.3 in IETF RFC 3986 [9]). In 3GPP APIs, the default apiRoot does not contain API Prefix (see e.g. Annex A.2 in 3GPP TS 29.502 [20]).

API URI: API URI has the following format: "{apiRoot}/<apiName>/<apiVersion>". For more details see clause 4.4.1. As an example, for the Nudm_SDM API defined in 3GPP TS 29.503 [28], the API URI is: "{apiRoot}/nudm-sdm/v2".

Base URI: Base URI is used as a reference against which the relative URI reference is applied. See clause 4.6.1.1.1.2 and also clause 5.1 in IETF RFC 3986 [9]. As an example, for the resource URI of the SM contexts collection that is defined in Nsmf_PDUSession API (see 3GPP TS 29.502 [20]), the base URI is "{apiRoot}/nsmf-pdusession/<apiVersion>".

Callback URI: Callback URI follows an absolute URI syntax, but excludes the following absolute URI identifiers: "userinfo" subcomponent of the "authority" component and also the "query" component ("fragment" component is already excluded from the absolute URI). Therefore, callback URI contains the "scheme", "authority" (excluding "userinfo" subcomponent) and the "path" components. See clause 4.4.3. As an example, for the Nsmf_PDUSession API defined in 3GPP TS 29.502 [20], "smContextStatusUri" is one of the callback references that can be provided by the NF Service Consumer. A complete callback URI for the "smContextStatusUri" has the following structure https://<comsumer-host>/<consumer-path-segment>/smContextStatus, where "consumer-host" is either an FQDN or an IP address (see clause 4.4.3).

Relative path after API URI: Refers to the sequence of path segments of a resource URI that are below the API URI (see the definition of the "Paths Object" in OpenAPI specification [4]). When appended to the API URI of the concerned API, it constitutes the resource URI. As an example, for the Nudm_SDM API defined in 3GPP TS 29.503 [28] for which the API URI is "{apiRoot}/nudm-sdm/v2", the "relative path after API URI" for the "AccessAndMobilitySubscriptionData" resource is: "/{supi}/am-data".

Relative URI: Relative URI is deprecated and shall not be use in SBI spcificationsSee clause 1.2.3 in IETF RFC 3986 [9].

Resource URI: Resource URI identifies an abstract or a physical resource. See the Abstract of the IETF RFC 3986 [9]. In this specification this generally means "an URI of a resource". Resource URI structure is defined in clause 4.4.1. As an example, for the resource URI of the SM contexts collection that is defined in Nsmf_PDUSession API (see 3GPP TS 29.502 [20]), the resource URI is "{apiRoot}/nsmf-pdusession/<apiVersion>/sm-contexts".

3.2 Abbreviations

For the purposes of the present document, the abbreviations given in 3GPP TR 21.905 [1] and the following apply. An abbreviation defined in the present document takes precedence over the definition of the same abbreviation, if any, in 3GPP TR 21.905 [1].

5GC	5G Core Network
CNF	Conjunctive Normal Form
CRUD	Create, Read, Update, Delete
DNF	Disjunctive Normal Form
HAL	Hypertext Application Language
HATEOAS	Hypermedia as the Engine of Application State
REST	REpresentational State Transfer
RPC	Remote-Procedure-Call
SBI	Service Based Interface
YAML	YAML Ain't Markup Language

3.3 Special characters, operators and delimiters

3.3.1 General

A number of characters have special meaning and are used as delimiters in this document and also in other stage 3 SBI specifications. Below clauses specify the usage of a selected set of the special characters. Full set of these special characters are specified in the respective IETF specifications.

3.3.2 ABNF operators

Special characters that are used as delimiters in ABNF syntax have somewhat different purpose from the same characters when used by URI syntax. See clause 3.3.2 in 3GPP TS 29.500 [2].

3.3.3 URI reserved and special characters

/   Reserved character. The forward slash character is a delimiter, which precedes an URI path component and also separates a sequence of path segments. See clauses 2.2 and 3.3 in IETF RFC 3986 [9].

\#   Reserved character. The number sign (hash) character is a delimiter, which terminates either an URI path component, or an authority component, or a query component and starts a fragment component. See clauses 2.2, 3.2, 3.3 and 3.5 in IETF RFC 3986 [9].

?   Reserved character. The question mark character is a delimiter, which terminates either an URI path component, or an authority component and starts a query component/parameter. See clauses 2.2, 3.2, 3.3 and 3.4 in IETF RFC 3986 [9].

=   Reserved character. The equal sign character is a delimiter, which separates a parameter name and its value, e.g. in an URI query. See clauses 1.2.3 and 3.3 in IETF RFC 3986 [9].

[ ]   Reserved character. The square bracket characters enclose an IPv6 literal address. This is the only place where square bracket characters are allowed in the URI syntax. See clause 3.2.2 in IETF RFC 3986 [9].

< >   Special characters. The angle bracket characters are delimiters and enclose a complete URI or an URI component or a subcomponent. See Appendix C in IETF RFC 3986 [9].

;   Reserved character. The semicolon character is a delimiter, which is often used to separates a parameter name and its value in a path segment. See clause 3.3 in IETF RFC 3986 [9].

:   Reserved character. The colon character is a delimiter, which either terminates an URI scheme component, or userinfo subcomponent, or separates host subcomponent from the port subcomponent, an authority component, or a query component and starts a fragment component. See clauses 2.2, and 3.2 in IETF RFC 3986 [9].

NOTE: The same characters, like "/", "#", etc. lead to different processing in ABNF and URI grammars. For instance, in URI syntax, ";" character separates parameter and its value, while in ABNF ";" starts a comment.

3.3.4 SBI specific usage of delimiters

The braces (curly brackets) are used as delimiters across the SBI specifications, but have somewhat different purpose as specified below.

{ }   Delimiters. The braces (curly brackets) characters enclose a name of a variable in an URI path segment (see clause 5.1.3.2). Example: …/subscriber-data/{supi}.

{ }   Delimiters. The braces (curly brackets) characters enclose a parameter value in a query (see clause 5.1.3.3). Example: ?nf-id={chooseAValue}

NOTE: In SBI specifications, "< >" is a generic placeholder, while "{ }" enclose specifically a variable.

4 Design Principles for 5GC SBI APIs

4.1 General Principles

Each 5GC SBI API specification should include the following information for each specified service:

- Purpose of the API;

- URIs of resources;

- Supported HTTP methods for a given resource;

- Supported representations (e.g. JSON, see IETF RFC 8259 [3]);

- Request body schema(s) (where applicable);

- Response body schema(s) (where applicable);

- Supported response status codes;

- Relation types supported if HATEOAS is implemented by the API;

- A reference in the resource description clause to one of the archetypes defined in Annex C if the resource design matches one of them; and

- A list defining identifiers of optional features (see clause 6.6 of 3GPP TS 29.500 [2] for related procedures).

For each specified service a clause to a normative Annex should be provided containing the OpenAPI definitions according to OpenAPI Specification [4] for the service. The specifications should state that content of this normative annex takes precedence when being discrepant to other parts of the specification with respect to the encoding of information elements and methods.

NOTE: The semantics and procedures, as well as conditions, e.g. for the applicability and allowed combinations of attributes or values, not expressed in the OpenAPI definitions but defined in other parts of the specification also apply.

The TS Skeleton Template as provided in Annex A should be used as a starting point when drafting 5GC SBI API specifications.

Common procedures, HTTP extensions and error handling applicable to several 5GC SBI API specifications should be defined in 3GPP TS 29.500 [2] and should be referenced from individual 5GC SBI API specifications.

Common data types applicable to several 5GC SBI API specifications should be defined in 3GPP TS 29.571 [5] and should be referenced from individual 5GC SBI API specifications.

4.2 API Design Style and REST Implementation Levels

4.2.1 General

5GC SBI API specifications should apply a protocol design framework as follows:

a) REST-style service operations should implement the Level 2 of the Richardson maturity model, with standard HTTP methods, whenever it is a good match for the style of interaction to model, e.g. service operations that can naturally map to one of the standard methods (CRUD operations), this should be the preferred modelling attempt;

b) service operations may use custom API operations (RPC-style interaction), when it is seen a better fit for the style of interaction to model, e.g. non-CRUD service operations;

c) it is possible to mix REST-style operations and RPC-style operations in the same API.

NOTE: Level 3 (HATEOAS) of the Richardson maturity model in the 5G Service-Based Architecture can be implemented by an API but is optional. Hypermedia usage guidelines are provided in clause 4.7 of the present specification.

4.2.2 API Design Principles for Query Operation

When designing a query operation API, i.e. the NF service consumer invokes the API aiming to retrieve certain information from the NF service producer, the following principles should be applied:

a) if the query operation does not require any input parameter for the NF service producer, then the REST-style service operation with standard HTTP GET method should be used (see clause 4.6.1.1.2);

b) if

- the query operation requires input parameter(s) for the NF service producer; and

- all the required input parameter(s) are used to identify a particular resource and/or control the content of the result of the query operation;

then the REST-style service operation with standard HTTP GET method should be used (see clause 4.6.1.1.2);

c) standard HTTP GET method shall not be used for non-safe operations and non-idempotent operations.

4.2.3 API Design Principles for Delete Operation

When designing a delete operation API, i.e. the NF service consumer invokes the API aiming to delete certain resource on the NF service producer, the following principles should be applied:

a) if the delete operation does not require any input parameter for the NF service producer, then the REST-style service operation with standard HTTP DELETE method should be used (see clause 4.6.1.1.4);

b) if

- the delete operation requires input parameter(s) for the NF service producer; and

- all the required input parameter(s) are used to identify a particular resource and/or control the content of the result of the delete operation;

then the REST-style service operation with standard HTTP DELETE method should be used (see clause 4.6.1.1.4);

c) standard HTTP DELETE method shall not be used for non-idempotent operations.

4.3   Version Control

4.3.0   General

The version control mechanism in the present clause allows the management of changes to an API and provides a version number that is incremented whenever changes to the API are applied.

>   NOTE: The version number does not reflect the usage of optional features. A mechanism to negotiate the usage of optional features is defined in clause 6.6 of 3GPP TS 29.500 [2].

4.3.1   Structure of API version numbers

4.3.1.1   API version number format

API version numbers shall consist of at least 3 fields, following a MAJOR.MINOR.PATCH pattern according to the Semantic Versioning Specification [17].

The 1st Field (MAJOR), the 2nd Field (MINOR), and the 3rd Field (PATCH) shall contain unsigned integer numbers, and they shall not contain leading zeroes.

An additional field (called "pre-release version" in Semantic Versioning Specification [17]) is added to denote an OpenAPI version under development, i.e. prior to the freeze of the corresponding OpenAPI description for a given 3GPP Release. This additional field is appended after the 3 first version fields using the hyphen "-" character and shall have the format "alpha.n", where "n" is an unsigned integer number without leading zeroes.

>   NOTE: An OpenAPI is referred to as "non-frozen", if for a given 3GPP Release the OpenAPI freeze milestone was not reached. Otherwise, an OpenAPI is referred to as "frozen".

After the freeze of a 3GPP Release, additional fields (called "build metadata" in Semantic Versioning Specification [17]), containing operator-specific version information, may be appended after the 3 first version fields using the plus sign "+" character and they shall consist of a list of dot-separated identifiers, where each identifier may contain only alphanumeric characters and hyphens ([0-9A-Za-z-]).

>   NOTE: Operator-specific version information are ignored when determining version precedence. Thus, two versions that differ only in the operator-specific version information, have the same precedence.

>   EXAMPLES:

>   "1.0.0-alpha.1"

>   "3.0.1+orange.2020-09"

4.3.1.2   Rules for incrementing field values

The first version of a new API under development shall obtain the version number "1.0.0-alpha.1". At the first publication of the 3GPP Technical Specification defining the API after the OpenAPI freeze of the first 3GPP Release that contains the API, the version number of the API shall be set to "1.0.0".

When a new version of the 3GPP TS containing OpenAPI file(s) is published, the fields of the corresponding API version number(s) shall be incremented according to the following rules:

-    The 1st Field (MAJOR) shall be incremented only if the applied change is backward incompatible relative to the earlier, i.e. frozen version of the given OpenAPI (see the details below). For a non-frozen OpenAPI, the first backwards incompatible change relative to the latest frozen version triggers incrementing the 1st Field (MAJOR), while subsequent backwards incompatible changes do not increment the value, until the OpenAPI stays non-frozen. When the 1st Field (MAJOR) is incremented, this impacts the lower level fields as follows:

    -    The 2nd Field (MINOR) shall be reset to "0";

    -    The 3rd Field (PATCH) shall be reset to "0";

- The 4th Field (NON-FROZEN), if present shall be reset to "alpha.1" (this field is removed when the OpenAPI freezes).

The 2nd Field (MINOR) shall be incrementd only if the applied change is a backward compatible new feature relative to the earlier, i.e. frozen version of the given OpenAPI (see the details below). For a non-frozen OpenAPI, the first backwards compatible change relative to the latest frozen version triggers incrementing the 2nd Field (MINOR), while subsequent backwards incompatible changes do not increment the value, until the OpenAPI stays non-frozen. When the 2nd Field (MINOR) is incremented, this impacts the lower level fields as follows:

 - The 3rd Field (PATCH) shall be reset to "0";

 - The 4th Field (NON-FROZEN), if present shall be reset to "alpha.1" (this field is removed when the OpenAPI freezes).

The 3rd Field (PATCH) shall be incrementd only if the applied change is a backward compatible correction relative to the earlier, i.e. frozen version of the given OpenAPI (see the details below). For a non-frozen OpenAPI, the first backwards compatible change relative to the latest frozen version triggers incrementing the 3rd Field (PATCH), while subsequent backwards incompatible changes do not increment the value, until the OpenAPI stays non-frozen. When the 3rd Field (PATCH) is incremented, this impacts the lower level field as follows:

 - The 4th Field (NON-FROZEN), if present shall be reset to "alpha.1" (this field is removed when the OpenAPI freezes).

- The 4th Field (NON-FROZEN), if present shall be incremented in a non-frozen OpenAPI every time any kind of changes are applied the given non-frozen OpenAPI (see the details below). This field is removed when the OpenAPI freezes.

1st Field (MAJOR):

- This numerical field shall be incremented when:

  a) there are one or more backward incompatible changes to the API after the OpenAPI freeze for a given 3GPP Release; and

  b) there are the first backward incompatible change(s) to the existing API with respect to the latest version in the previous 3GPP Release while a 3GPP Release is under development (i.e. prior to the OpenAPI freeze for a given 3GPP Release).

EXAMPLE 1:	Assuming that 3GPP Rel-16 under development contains API version "1.1.0-alpha.2", and a backward incompatible change with respect to the latest version in the previous 3GPP Release is applied to that API before the OpenAPI freeze, the new Rel-16 API version is "2.0.0-alpha.1".

NOTE 1:	Subsequent changes in a given 3GPP Release under development do not lead to increment of the 1st field (MAJOR) and 2nd field (MINOR).

NOTE 2:	Rules for determining backward incompatible changes are provided in Annex B.

NOTE 3:	It is recommended to avoid backward incompatible change to the API after the OpenAPI freeze whenever possible, especially after OpenAPI freeze of a succeeding Release. It is preferable to introduce such changes only in the 3GPP Release under development.

- If a backward incompatible change needs to be applied to several 3GPP Releases the following applies:

  a) If the 3GPP Releases contain different MAJOR versions of the same API, a new MAJOR API version shall be assigned to each 3GPP Release in the order of those 3GPP Releases in such a manner that the lowest of those 3GPP Releases shall obtain the first unassigned MAJOR version value.

EXAMPLE 2:	Assuming that 3GPP Rel-15 contains API version "1.0.0", and Rel-16 contains API version "2.0.0", and that the same backward incompatible change is applied to that API in both Releases, the new Rel-15 API version is "3.0.0" and the new Rel-16 API version is "4.0.0".

  b) If the 3GPP Releases contain the same MAJOR version but different MINOR versions of the same API, a single new MAJOR API version value shall be assigned for all those 3GPP Releases, unless other

backward incompatible changes only applied to some of those Releases require the creation of separate MAJOR versions.

NOTE 4: For each such Release a new MINOR version is assigned.

EXAMPLE 3: Assuming that 3GPP Rel-15 and Rel-16 contain API version "1.0.0", and Rel-17 contains API version "1.2.0", and that the same backward incompatible change is applied to that API in all 3GPP Releases, the new 3GPP Rel-15 and Rel-16 API version is "2.0.0" and the new 3GPP Rel-17 API version is "2.2.0".

    c) If the 3GPP Releases contain the same API versions, a single new API version shall be assigned for all those 3GPP Releases, unless other changes only applied to some of those Releases require the creation of separate versions.

EXAMPLE 4: Assuming that 3GPP Rel-15 and 3GPP Rel-16 contain API version "1.0.0", and that only the same backward incompatible change is applied to that API in both 3GPP Releases, the new 3GPP Rel-15 and Rel-16 API version is "2.0.0".

EXAMPLE 5: Assuming that 3GPP Rel-15 and Rel-16 contain API version "1.0.0", and that the same backward incompatible change is applied to that API in both Releases and an additional backward compatible change is applied in 3GPP Rel-16, the new 3GPP Rel-15 API version is "2.0.0", and the 3GPP Rel-16 API version is "2.1.0".

EXAMPLE 6: Assuming that 3GPP Rel-15 and Rel-16 contain API version "1.0.0", and that the same backward incompatible change is applied to that API in both Releases and an additional backward incompatible change is applied in 3GPP Rel-16, the new 3GPP Rel-15 API version is "2.0.0", and the 3GPP Rel-16 API version is "3.0.0".

2nd Field (MINOR):

- This numerical field shall be incremented when:

    a) there are the first one or more backward compatible changes not corresponding to changes to earlier 3GPP Releases (i.e. changes introduced by 3GPP CR with other categories than "mirror") to the same API in a given 3GPP Release without any prior backward incompatible changes in that Release. If the same 1st field (MAJOR) and the 2nd field (MINOR) are assigned to n previous 3GPP Releases, a MINOR version number shall be reserved for each intermediate 3GPP Release for possible subsequent changes in that Release and the MINOR version number shall be incremented by n; and

EXAMPLE 7: Assuming that 3GPP Rel-15 and Rel-16 contain API version "1.0.0" (because there were no changes to the API in Rel-16), and in Rel-17 the first backward compatible new feature is added before the OpenAPI freeze, the API version "1.2.0-alpha.1" is assigned to Rel-17.

    b) there are one or more subsequent backward compatible additions of features not corresponding to changes to previous 3GPP Releases to the API in a frozen 3GPP Release before a higher MINOR number has been allocated for the same MAJOR version (for a subsequent Release).

- This field shall be reset to "0" if the 1st field (MAJOR) is changed, unless a backward incompatible change needs to be applied to several 3GPP Releases that already contain the same MAJOR but different MINOR API versions. In that case a single new major API version is assigned, and for each such 3GPP Release with an own MINOR version, a new MINOR version shall be assigned, starting with MINOR version "0" for the lowest such Release, and reserving a MINOR version number for each intermediate Release without an own MINOR version. (see Example 3)

NOTE 5: In most cases the MINOR version is incremented when new backward compatible features are added in a 3GPP Release. In rare cases, where only backward compatible changes not corresponding to changes to previous 3GPP Releases are applied to a 3GPP Release, the MINOR version is also incremented. It is recommended to avoid such changes in 3GPP Releases without added functionality whenever possible.

NOTE 6: Subsequent backward compatible changes in a given 3GPP Release before OpenAPI freeze do not lead to an increment of the 2nd field (MINOR).

NOTE 7: Changes corresponding to changes in previous 3GPP Releases do not lead to an increment of the 2nd field (MINOR).

NOTE 8: If two 3GPP Releases are under parallel development (because the work on Rel-*X+1* has commenced before the OpenAPI freeze of Rel-*X*), the corresponding APIs will obtain distinct values of the 1st field (MAJOR) or 2nd field (MINOR).

EXAMPLE 8: Assuming that an API was introduced with version "1.0.0" in Rel-15, and that the Rel-16 version is "1.1.0-alpha.5" because the OpenAPI is not yet frozen in Rel-16, and that a new backward compatible Rel-17 feature is added, the Rel-17 API version is "1.2.0-alpha.1".

3rd Field (PATCH):

- This numerical field shall be incremented:

 a) if the changes are only one or more backward-compatible corrections (but no changes requiring an update of the 1st field (MAJOR) or of the 2nd field (MINOR) are made to the API after the OpenAPI freeze of a 3GPP Release; and

 b) if one or more backward compatible additions of features, but no changes requiring an update of the 1st field (MAJOR) or of the 2nd field (MINOR), are made to the API after the OpenAPI freeze of a 3GPP Release and after the assignment of a MINOR version to a higher 3GPP Release.

- This field shall be reset to "0" if the 1st field (MAJOR) or 2nd field (MINOR) is changed.

NOTE 9: Before the OpenAPI freeze for a given 3GPP Release, the 3rd field (PATCH) will not be incremented.

NOTE 10: If the 1st field (MAJOR) and 2nd field (MINOR) were not incremented between 3GPP Releases (because there were no added features and no backward incompatible changes), and the same backward compatible changes are then applied to those 3GPP Releases, the API files in those 3GPP Releases are identical and will obtain the same API version number.

NOTE 11: In rare cases for which a new backward compatible functionality needs to be added in an older 3GPP Release after the OpenAPI freeze and work on that API already started in a later Release, the new functionality is exceptionally introduced as a PATCH correction and a new supported feature could be defined accordingly.

The 4th Field (NON-FROZEN and/or OPERATOR-SPECIFIC):

- The 4th Fields may or may not be present, based on the following conditions.

- NON-FROZEN type. The presence of this filed type depends on the OpenAPI status, as follows:

 - The 4th Field (NON-FROZEN) shall be present before the OpenAPI freeze of a 3GPP Release. An additional field (separated from the 3 first version fields by a hyphen "-" character) shall be supplied as follows:

 a) When the 1st field (MAJOR) or 2nd field (MINOR) is incremented before the OpenAPI freeze of a 3GPP Release, this field shall obtain the value "alpha.1".

 b) The numerical value "*n*" within the field value "alpha.*n*" shall be incremented if one or more subsequent changes (either backward incompatible, or backward compatible) are made to the given non-frozen OpenAPI.

 - The 4th Field (NON-FROZEN) shall not be present after the OpenAPI freeze of a 3GPP Release., this additional field (including the hyphen "-" character) containing pre-release version info is removed from the API version.

- OPERATOR-SPECIFIC type that represents operator-specific version field(s) . The presence of these fields depends on the OpenAPI status, and the operator preferences:

 - The 4th Field (OPERATOR-SPECIFIC) shall be present after the OpenAPI freeze of a 3GPP Release. Additional fields may be supplied based on operator policy after the 3 first version fields and separated by a plus sign "+" character. The rules for setting, or incrementing, such fields are out of the scope of 3GPP, but they shall comply with the rules described in the Semantic Versioning Specification [17], and contain a list of dot-separated labels within the allowed character set ([0-9A-Za-z-]).

If no change is applied to an API in a new published TS version, the API version number shall not be incremented unless the additional field ("-alpha.*n*") needs to be removed at OpenAPI freeze. This also applies if the TS is published in a new 3GPP Release.

NOTE 12: OpenAPI files can contain references to other OpenAPI files. Changes to referenced parts of such other OpenAPI files need to be considered when determining if and how to update an API version.

NOTE 13: The API version number of those version fields managed by 3GPP is incremented using 3GPP change requests.

4.3.1.3 Visibility of the API version number fields

The API version shall be indicated in the resource URI of every API, as described in clause 4.4.1.

The API version shall be indicated as the concatenation of the letter "v" and the 1st field (MAJOR) of the API version number.

The other fields shall not be included in the resource URI.

NOTE: Including these digits in the URI would force the NF service consumer to select a specific sub-version, at the risk of seeing the request rejected if the NF service provider does not support it, while the request could have been served by ignoring unknown elements.

The full API version number (i.e., containing all the fields) shall be visible in the OpenAPI specifications, in the "version" subfield of the "info" field, as described in clause 5.3.3.

4.3.1.4 Relation to the Technical Specification version number

There is no one-to-one mapping between an API version number and the version number of the 3GPP Technical Specification defining this API.

A 3GPP Technical Specification specifies one or more APIs, which may have different versions.

A change in the 3rd field (PATCH) of a 3GPP TS version number (i.e. an editorial change) should not lead to a change in the version number of the APIs specified in the 3GPP TS.

A change in the 1st field (MAJOR) and 2nd fields (MINOR) of the 3GPP TS version number is likely to lead to at least a change in the minor version number of the APIs specified in the 3GPP TS.

EXAMPLE: If version 15.4.1 of a 3GPP TS contains version "1.1.1" of API A, B and C, and a version 16.0.0 of this 3GPP TS is derived from version 15.4.1, TS version 16.0.0 can contain version "1.2.0.alpha-1" of API A (if all changes made are backward compatible), version "2.0.0.alpha-1" of API B (if some changes are no backward compatible) and version "1.1.1" of API C (if no changes were made).

The 3GPP TS defining the API is indicated in the OpenAPI specification of the API, as described in clause 5.3.4.

4.3.1.5 Discovery of the supported versions

The NF service consumer may discover the API version(s) supported by an NF service producer using the following mechanisms:

- NRF query: The NF service consumer may retrieve from the NRF the NF profile of a given NF Instance. This NF profile contains the full version number(s) of the API(s) supported by an NF Service Instance, as described in the clause 6.2.6.2.4 of 3GPP TS 29.510 [18] and the planned retirement date.

- NF profile change notifications: The NF service consumer may subscribe for NF status change notifications with the NRF as specified in clause 5.2.2.5 of 3GPP TS 29.510 [18]. The NRF shall notify as specified in clause 5.2.2.6 of 3GPP TS 29.510 [18], any change to the NF profile which may include updated NF service profile containing the current list of NF services and their versions supported by the NF.

When a new major version is created, the NF service producer shall continue supporting at least the previous major version until a retirement date unless all API versions (except for draft API versions published prior to the OpenAPI freeze) with that previous major version are withdrawn (see clause 4.3.1.6); this enables NF service consumers to

migrate to the new version. After expiration of the retirement date, the old major version should be deprecated. The retirement date of an old major version supported by a NF service instance may be updated in the NF profile in the NRF.

4.3.1.6   Withdrawing API versions

If it is discovered that one or several previous API versions are not providing the basic mandatory functionality of an API due to severe functional or encoding deficits (for instance, there is no or very limited interoperability between the NF service consumer and NF service producer when such an API version is used, or the API is hardly implementable because of severe deficits in the OpenAPI file that cannot easily be fixed by implementers in an interoperable manner), those API versions shall be listed as withdrawn in subsequent versions of the TS defining the corresponding API; any withdrawn API versions from the same or previous 3GPP releases shall be listed. API versions published before the OpenAPI freeze of the corresponding 3GPP Release, i.e. with a 4th Field (NON-FROZEN and/or OPERATOR-SPECIFIC) as part of the version number, shall not be withdrawn.

> NOTE 1: It is recommended to avoid withdrawing API versions whenever possible. It is expected that a need to withdraw API versions is most likely detected when discussing corrections soon after the OpenAPI freeze of a new API.

> NOTE 2: Corrections to optional or minor parts of the API functionality do not lead to the withdrawal of API versions. However, if severe functional or encoding deficits of the functionality related to an optional functionality with a corresponding supported feature (see 3GPP TS 29.500 [2] clause 6.6.2) are discovered, a new supported feature can be introduced to enable a negotiation of the support of the correction, and the old corresponding supported feature can be marked as "withdrawn" in the table defining the supported features of an API.

Withdrawn API versions should not be deployed.

4.4   URI Structure

4.4.1   Resource URI structure

Resources are either individual resources, or structured resources that can contain child resources. It is recommended to design each resource following one of the archetypes provided in the Annex C.

A URI uniquely identifies a resource. In the 5GC SBI APIs, when a resource URI is an absolute URI, its structure shall be specified as follows:

> **{apiRoot}/<apiName>/<apiVersion>/<apiSpecificResourceUriPart>**

"apiRoot" shall be a concatenation of the following parts:

- scheme ("http" or "https")

> NOTE: In this release of the specification both http and https scheme URIs are allowed. See clause 13.1 of 3GPP TS 33.501[22] for further details on security of Service Based Interfaces.

- the fixed string "://"

- authority (host and optional port) as defined in IETF RFC 3986 [9]. The host should be represented by the operator specific FQDN (for FQDN examples see clause 28.3.2 in 3GPP TS 23.003 [27]).

- an optional deployment-specific string (API prefix) that starts with a "/" character.

apiName" shall define the name of the API. For example, 'nsmf-pdusession'.

"apiVersion" shall indicate the MAJOR version of the API. The format is specified in clause 4.3.1.3. For example, 'v1'.

While "apiRoot", "apiName" and "apiVersion" together define the API URI of the corresponding API, each "apiSpecificResourceUriPart" defines a resource URI of the API relative to the API URI.

An API URI should not contain a trailing slash as defined in clause 3.1, and if it contains one, then it should be ignored/removed.

Example of an API URI from 3GPP TS 29.502 [20]:

 - url: '{apiRoot}/nsmf-pdusession/v1'

4.4.2 Custom operations URI structure

The custom operation definition is in Annex C.

The URI of a custom operation which is associated with a resource shall have the following structure:

> **{apiRoot}/<apiName>/<apiVersion>/<apiSpecificResourceUriPart>/<custOpName>**

Custom operations can also be associated with the service instead of a resource. The URI of a custom operation which is not associated with a resource shall have the following structure:

> **{apiRoot}/<apiName>/<apiVersion>/<custOpName>**

In the above URI structures, "apiRoot", "apiName", "apiVersion" and "apiSpecificResourceUriPart" are as defined in clause 4.4.1 and "custOpName" represents the name of the custom operation as defined in clause 5.1.3.2.

4.4.3 Callback URI structure

The purpose of the callback URI is to enable NF service consumer to provide the URI to be used by an NF Service Producer to send notification or callback requests.

The callback URI shall be in the form of an absolute URI as defined in clause 4.3 of IETF RFC 3986 [9], including an authority, and excluding any query component, any fragment component and any userinfo subcomponent.

Therefore, callback URI consists of the following components, specified with ABNF syntax (see IETF RFC 5234 [26]):

URI = scheme ":" "//" host [":" port] [prefix] "/" <callbackUriPart>

Where:

- scheme is either "http" or "https";

  NOTE: In this release of the specification both http and https scheme URIs are allowed. See clause 13.1 of 3GPP TS 33.501[22] for further details on security of Service Based Interfaces.

- 'host' is either an FQDN or an IP address;

- the URI may contain an optional port;

- the URI may contain an optional deployment-specific string (prefix) that starts with a "/" character;

- the "callbackUripart" is the part of the callback URI, after the prefix (if any), of the path to an HTTP resource on the consumer, which is implementation specific; and

- the path of the callback URI comprises the prefix (if any) and the "callbackUriPart".4.5 Resource Representation and Content Format Negotiation

4.5.1 Resource Representation

A resource representation is a serialization of the resource state in a particular content format. It's included in the data frame of an HTTP/2 request or response. Representation header fields provide metadata about the representation. When a message includes a data frame, the representation data enclosed in the data frame. HTTP/2 reuses the definition of Representation header as HTTP 1.1 in IETF RFC 9110 [6]. Content-type field in HTTP/2 header performs as representation header fields and describes the representation data that would have been enclosed in the data frame, e.g. if content-type is application/json, resource representation in data frame is serialized in JSON format.

Server supports the content format of the representation received in the data frame of the request and returns the "200 OK" response code.

4.5.2 Content Format Negotiation

IETF RFC 9110 [6] provides a mechanism to negotiate the content format of a representation.

In HTTP/2 requests and responses, the "Content-Type" HTTP/2 header field is used to signal the format of the actual representation included in the data frame. If the format of the representation in an HTTP/2 request is not supported by the server, it responds with the "415 Unsupported Media Type" response code.

For GET method, the "Accept" HTTP header of the HTTP/2 request signals the content formats that a client supports. If the server cannot provide any of the accepted formats, it returns the "406 Not Acceptable" response code.

4.6 Use of HTTP Methods

4.6.1 Use of Request/Response Communication

4.6.1.1 CRUD

4.6.1.1.1 Creating a Resource

4.6.1.1.1.1 General

Procedures that allow an NF service consumer to create a new resource at the NF service producer shall be specified to either use the HTTP POST method with procedures according to clause 4.6.1.1.1.2 or the HTTP PUT method with procedures according to clause 4.6.1.1.1.3.

4.6.1.1.1.2 Creating a Resource using POST

The HTTP POST method (see IETF RFC 9110 [6]) allows an NF service consumer to create a new child resource at the NF service producer in such a manner that the NF service producer selects the child resource identifier and the URI for the child resource.

Figure 4.6.1.1.1.2-1 illustrates creating a resource using POST.
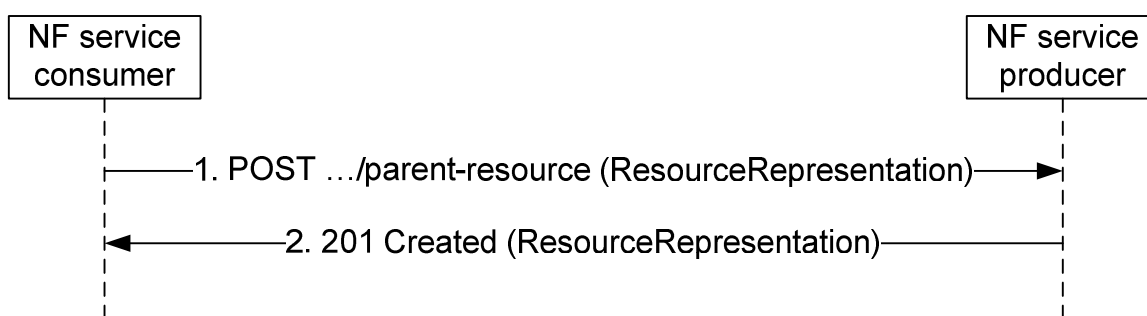
Figure 4.6.1.1.1.2-1: Creating a resource using POST

1. The parent resource of which the new resource is to be created as a child is identified by the request URI. The content of the POST request shall contain a representation of the resource to be created without a child resource identifier. For forward compatibility, the NF service producer ignores unknown attributes in the received resource representation unless specified otherwise by the particular application.

2. The NF service producer generates a child resource identifier and constructs the URI for the created resource by appending that child resource identifier to the parent resource URI received as request URI of the POST request (e.g. "…/parent-resource/childresource1").

   On success, "201 Created" shall be returned, the content of the POST response should contain a representation of the created resource, and the "Location" header shall be present and shall contain the URI of the created resource. The created resource shall be served by the same NF (service) instance that received the service request, unless the 5GC SBI API specifications explicitly specified that in specific use cases the created resource

may be served by another NF (service) instance. If in such specific use cases the resource is created in a different NF (service) instance, the identifier of the serving NF (service) instance shall be included in the response message.

If the HTTP scheme used in the returned URI is "https", then the authority of the URI included in the Location header shall be an FQDN, and not an IP address.

The URI included in the "Location" header may be an absolute URI or a relative URI reference (see IETF RFC 3986 [9]); when the URI is in relative form, the base URI used to resolve the URI reference is the target URI included in the received POST request.

NOTE: The representations of the resource in the request and response can differ, e.g. the representation of the resource in the response can be empty or can contain a subset of the representation as received in the request possibly with modified attributes, and in addition can contain additional attributes. Exact details will be specified by the application.

On failure, the appropriate HTTP status code indicating the error shall be returned and appropriate additional error information should be returned in the POST response body (see clause 4.8).

A collection may be used to model a resource that serves as a directory of resources that may be distributed on different processing instances or hosts. If so:

- the authority and/or deployment-specific string of the apiRoot of the created resource URI returned by the NF Service Producer in the "Location" header may differ from the authority and/or deployment-specific string of the apiRoot of the request URI received in the POST request.

- the NF Service Consumer shall be capable to receive and process an authority and/or deployment-specific string in the apiRoot of the created resource URI that differs from the authority and/or deployment-specific string of the apiRoot of the Request URI.

It needs to be clearly stated in the 5GC SBI API specifications when a NF Service Producer may return a different authority and/or deployment-specific string in the apiRoot of the created resource URI for a collection resource.

4.6.1.1.1.3 Creating a Resource using PUT

The HTTP PUT method (see IETF RFC 9110 [6]) allows an NF service consumer to create a new resource at the NF service producer in such a manner that the NF service consumer selects the resource identifier and the URI for the resource.

Figure 4.6.1.1.1.3-1 illustrates creating a resource using HTTP PUT.
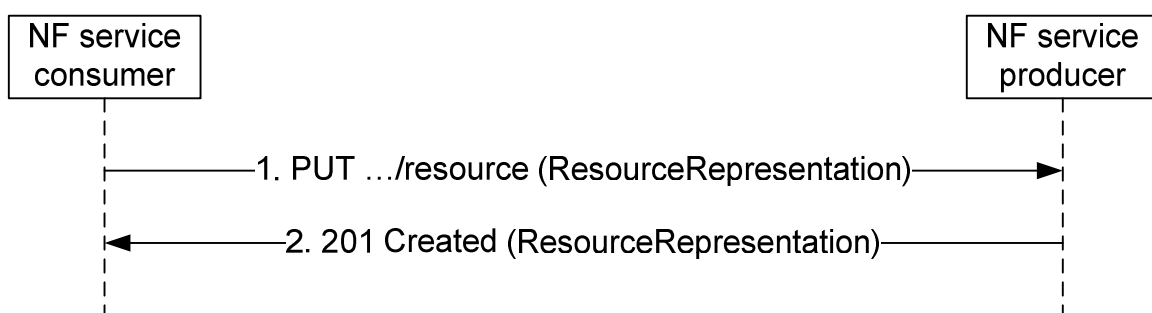
Figure 4.6.1.1.1.3-1: Creating a Resource using HTTP PUT

1. The NF service consumer selects a resource identifier and constructs the URI for the resource to be created by appending that resource identifier to the parent resource URI. The resource that is to be created is identified by that URI as request URI. The content of the PUT request shall contain a representation of the resource to be created. For forward compatibility, the NF service producer ignores unknown attributes in the received resource representation unless specified otherwise by the particular application.

2. On success, "201 Created" shall be returned, the content of the PUT response should contain the representation of the created resource, and the "Location" header shall be present and shall contain the URI of the created resource. The created resource shall be served by the same NF (service) instance that received the service

request, unless the 5GC SBI API specifications explicitly specified that in specific use cases the created resource may be served by another NF (service) instance. If in such specific use cases the resource is created in a different NF (service) instance, the identifier of the serving NF (service) instance shall be included in the response message.

If the HTTP scheme used in the returned URI is "https", then the authority of the URI included in the Location header shall be an FQDN, and not an IP address.

NOTE: The representations of the resource in the request and response can differ, e.g. the representation of the resource in the response can be empty or can contain a subset of the representation as received in the request possibly with modified attributes, and in addition can contain additional attributes. Exact details will be specified by the application.

On failure, the appropriate HTTP status code indicating the error shall be returned and appropriate additional error information should be returned in the PUT response body (see clause 4.8).

If the resource that is to be created already exists at the NF service producer, the following applies:

1) If the update of that resource by PUT is supported, the existing representation of the resource is replaced with the representation received in the PUT request body; see clause 4.6.1.1.3.1.

2) If the update of that resource by PUT is not supported, the "403 Forbidden" HTTP status code shall be returned and appropriate additional error information should be returned in the PUT response body (see clause 4.8).

4.6.1.1.2 Reading a Resource

4.6.1.1.2.1 Reading a Single Resource

Procedures that allow a service consumer NF (client) to read information from the server shall be specified to use the HTTP GET method (see IETF RFC 9110 [6]) to obtain the current representation of a resource.

Figure 4.6.1.1.2-1 illustrates reading a resource.
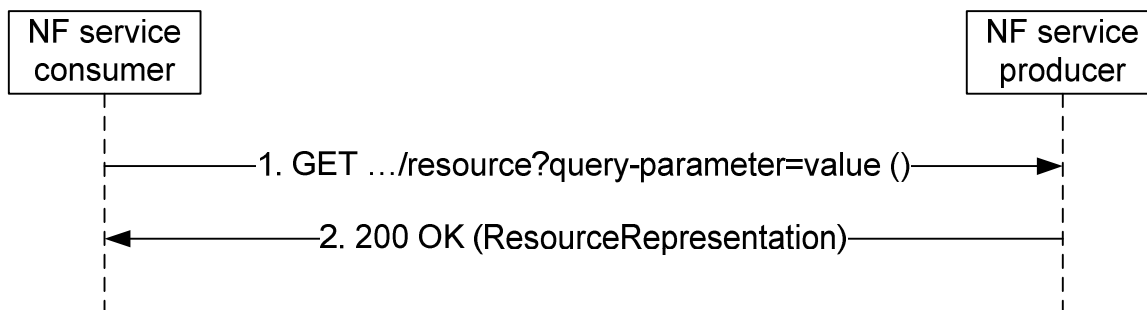
Figure 4.6.1.1.2.1-1: Reading a resource

1. The resource of which a representation is to be obtained is identified by the request URI. Query parameters may be used to control the content of the result.

The content of the GET request shall be empty.

2. On success, "200 OK" shall be returned and the content of the GET response shall contain the obtained resource representation.

On failure, the appropriate HTTP status code indicating the error shall be returned and appropriate additional error information should be returned in the GET response body (see clause 4.8).

4.6.1.1.2.2 Querying a Set of Resources

Procedures that allow a service consumer NF (client) to querying a set of resources from the server shall be specified to use HTTP GET method towards a resource modelled as Collection or Store archetype.

Query parameters (see clause 4.6.1.1.5) may be provided when querying a set of resources. The query component contains non-hierarchical data that, along with data in the path component, to filter the resources identified within the scope of the URI's scheme to a subset of the resources matching the query parameters. The query component is indicated by the first question mark ("?") character and terminated by a number sign ("#") character or by the end of the URI.
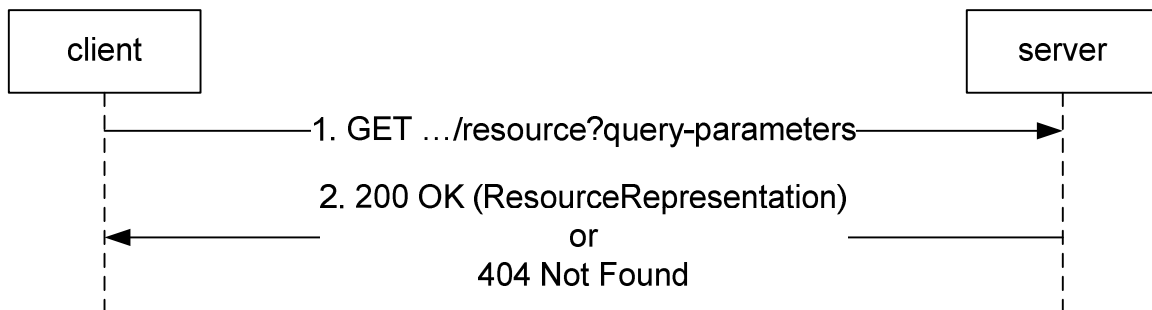
Figure 4.6.1.1.2.2-1: Query of a collection of resources by using query parameters.

Step 1. The client shall send a HTTP GET request using the URI of a resource modelled as Collection or Store archetype, optionally with query parameters, to the server.

Step 2. On success, the server shall return a set of sub-resources that includes only those entries filtered by the query parameters. If no sub-resource is matched for the querying service operation, the server shall return "200 OK" with an empty array (e.g. "[ ]" in JSON) in response body. If the resource in the URI doesn't exist on the server, the server shall return "404 Not Found" with optionally the cause information in response body.

NOTE: The result array/empty array can be defined as an attribute of an object, if the service operation returns an object in the response content for extensibility consideration.

Clause 4.9 specifies some possible options for an NF Service Producer to return the representations of multiple resources to a NF Service Consumer.

4.6.1.1.3 Updating a Resource

4.6.1.1.3.1 Usage of HTTP PUT

Procedures that allow a service consumer NF (client) to update information stored at the server by means of a complete replacement shall be specified to use the HTTP PUT method to replace the current representation of a resource with a new representation.

Figure 4.6.1.1.3.1-1 illustrates updating a resource using HTTP PUT.
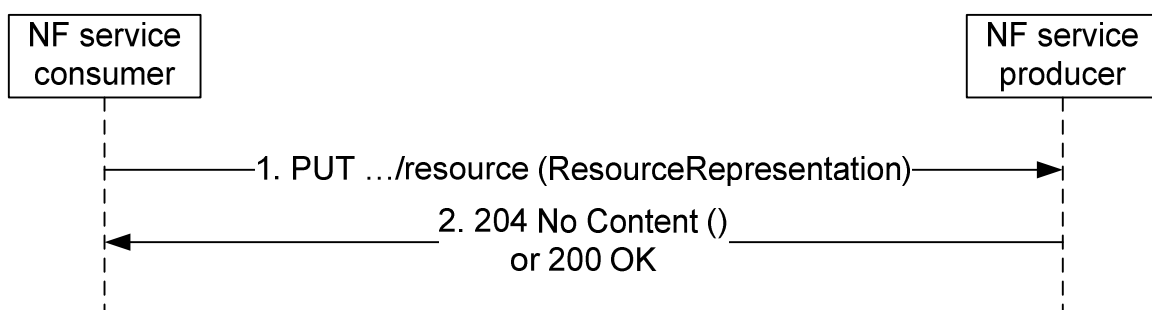
Figure 4.6.1.1.3.1-1: Updating a Resource using HTTP PUT

1. The resource that is to be updated is identified by the request URI. The content of the PUT request shall contain the new representation of the resource. For forward compatibility, the NF service producer ignores unknown attributes in the received resource representation unless specified otherwise by the particular application.

2. On success, "204 No Content" or "200 OK" shall be returned.

On failure, the appropriate HTTP status code indicating the error shall be returned and appropriate additional error information should be returned in the PUT response body (see clause 4.8).

If the resource that is to be updated does not exist at the NF service producer, the following applies:

1. If the creation of that resource by PUT is supported, the resource is created according to the procedure in clause 4.6.1.1.1.3.

2. If the creation of that resource by PUT is not supported, the "403 Forbidden" HTTP status code shall be returned and appropriate additional error information should be returned in the PUT response body (see clause 4.8).

4.6.1.1.3.2 Usage of HTTP PATCH

Procedures that allow a service consumer NF (client) to update information stored at the server by means of a partial replacement shall be specified to use the HTTP PATCH method (see IETF RFC 5789 [10]) to modify the current representation of a resource according to given modification instructions. The format of the PATCH message body shall be specified for each resource where the PATCH method is supported using one or several of the following encodings:

- If no modification of individual elements within an array needs to be supported, the "JSON Merge Patch" encoding of changes defined in IETF RFC 7396 [7] should be used.

- If a modification of individual elements within an array needs to be supported, the "JSON Patch" encoding of changes defined in IETF RFC 6902 [8] shall be used.

- If a modification of a resource with multiple datasets needs to be supported, the HTTP multipart message with the multipart/mixed content-type as described in IETF RFC 2046 [29] shall be used.

A single of the above encodings shall be specified for each resource where the PATCH method is supported unless backward compatibility considerations necessitate the support of both encodings.

NOTE 1: In Rel-15 a single encoding will be selected for each resource as backward compatibility considerations do not yet apply.

NOTE 2: "JSON Merge Patch" does not support the modification of individual elements within an array. However, it supports the modification of individual elements within maps (see clause 5.2.4.2). Collections of elements can be modelled as maps, instead of arrays, if a partial modification using PATCH is desired.

NOTE 3: The Open API description of the body of HTTP PATCH requests is specified in clause 5.3.8.

Figure 4.6.1.1.3.2-1 illustrates updating a resource using HTTP PATCH.
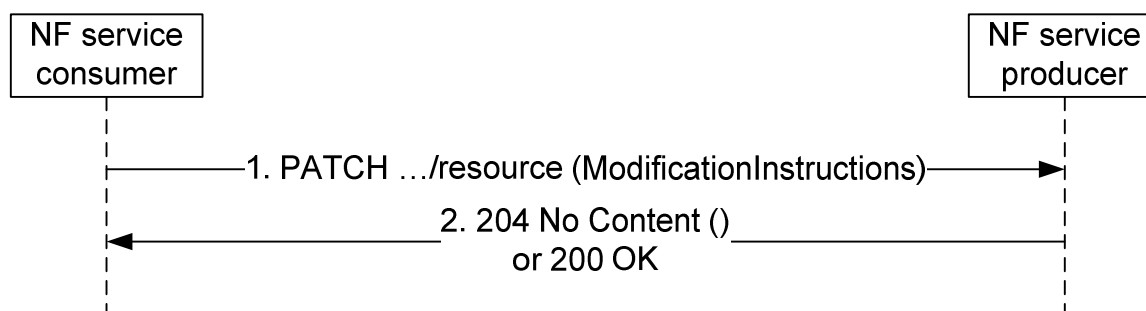
Figure 4.6.1.1.3.2-1: Updating a Resource using HTTP PATCH

1. The resource that is to be updated is identified by the request URI. The content of the PATCH request shall contain a description of the requested modifications of the resource. For the "JSON Merge Patch" encoding defined in IETF RFC 7396 [7] and the "Content-Type" header shall be set to "application/merge-patch+json". For the "JSON Patch" encoding of changes defined in IETF RFC 6902 [8] the "Content-Type" header shall be set to "application/json-patch+json". For resources with multiple datasets the encoding for HTTP multipart messages with the multipart/mixed content-type as defined in IETF RFC 2046 [29] shall be used. For forward compatibility, the NF service producer shall ignore received modification instructions of unknown attributes in the resource unless specified otherwise by the particular application.

2. On success, "204 No Content" or "200 OK" shall be returned.

On failure, the appropriate HTTP status code indicating the error shall be returned and appropriate additional error information should be returned in the PATCH response body (see clause 4.8).

4.6.1.1.4 Deleting a Resource

Procedures that allow a service consumer NF (client) to delete a resource from the server shall be specified to use the HTTP DELETE method (see IETF RFC 7231 [6]).

Figure 4.6.1.1.4-1 illustrates deleting a resource.
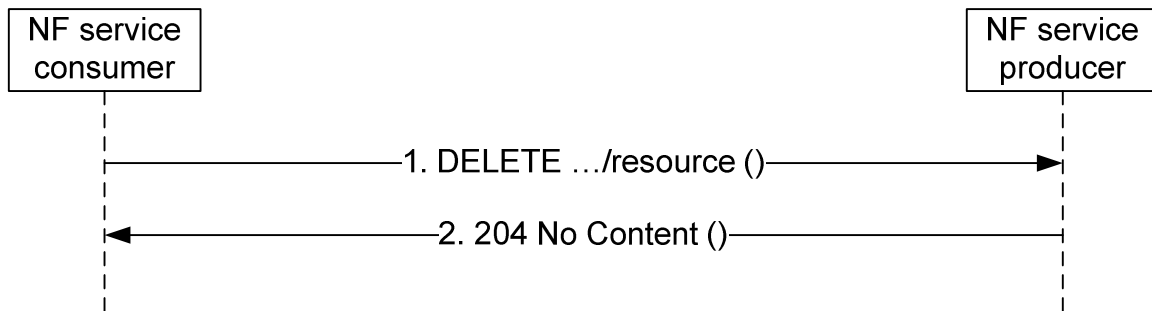
Figure 4.6.1.1.4-1: Deleting a resource

The resource that is to be deleted is identified by the request URI.

The payload body of the DELETE request shall be empty.

On success, "204 No Content" should be returned and then the payload body of the DELETE response shall be empty.

On failure, the appropriate HTTP status code indicating the error shall be returned and appropriate additional error information should be returned in the DELETE response body (see clause 4.8).

4.6.1.1.5 Query Parameters

4.6.1.1.5.1 General

The optional query component in the URI contains non-hierarchical data that, along with data in the path component, serves to filter the resources identified within the scope of the URI's scheme to a subset of the resources matching the query parameters. The query component is indicated by the first question mark ("?") character and terminated by a number sign ("#") character or by the end of the URI. Absence of the query component from the URI is indicated by absence of a question mark character from the URI. The syntax of the query component is specified in IETF RFC 3986 [9].

When a server receives a request with a query component, it shall parse the query string in order to identify filters. The first question mark is used to be a separator and is not part of the query string. A query string is composed of a possibly empty series of "key=value" pairs (also referred to as series of query parameters), separated by "&". If one query parameter contains more than one value, i.e. an array of data elements, different values shall be separated by comma (",").

The behaviour of the server, when receiving an HTTP/2 method with a query parameter which is of type array and only some of the members in the array can be matched, depends on each API and the behaviour shall be clearly described.

If multiple query parameters are defined for a method on the resource, the logical 'AND' represents the default logical relationship between the query parameters for this resource. If a logical relationship between multiple query parameters is specified in an API, then this overrides the default relationship. If multiple query parameters are defined for a method on the resource in an API, but there is no need to specify any logical relationship between these query parameters, the API shall explicitly state this.

4.6.1.1.5.2 Complex query expression

The complex query expression is used when there are multiple query parameters in the URI and the query condition needs to be expressed by a logical combination of multiple query parameters which overrides the default logical relationship of the query parameters. The complex query expression is either a Conjunctive Normal Form (CNF) or a Disjunctive Normal Form (DNF) which is equivalent to the logical combination of query parameters reflecting the query condition.

The "complex-query" query parameter may be used when a complex query expression is needed to express a query condition. The value of the "complex-query" query parameter is of type "ComplexQuery" which is a JSON object, the corresponding CNF or DNF is encoded into that JSON object (see 3GPP TS 29.571 [5] for the details of the data type "ComplexQuery"). The use of "complex-query" shall be negotiated using the feature negotiation procedure as defined in 3GPP TS 29.500 [2].

If a query parameter is included in the "complexQuery" then the same query parameter shall not be included outside the "complexQuery" in the same request message.

NOTE 1: It is not assumed that all APIs support "complex-query", the API supports this feature only when it is described in the corresponding specification.

NOTE 2: The logical relationship between "complex-query" and the other query parameters defined for a particular API is described in the corresponding specification of that API.

NTOE 3: The "complex-query" is not an additional explanation of the other query parameters, the condition expressed in the "complex-query" is evaluated along with the other queries.

4.6.1.2 Custom Operations

Custom Operations provide procedures that allow a service consumer NF (client) to interact with an NF service producer in other ways than what is supported by the CRUD methods described in clause 4.6.1.1.

Custom Operation can be related to a resource or can be related to an entire service and be independent of a resource.

Figure 4.6.1.2-1 illustrates the use of a custom operation related to a resource.
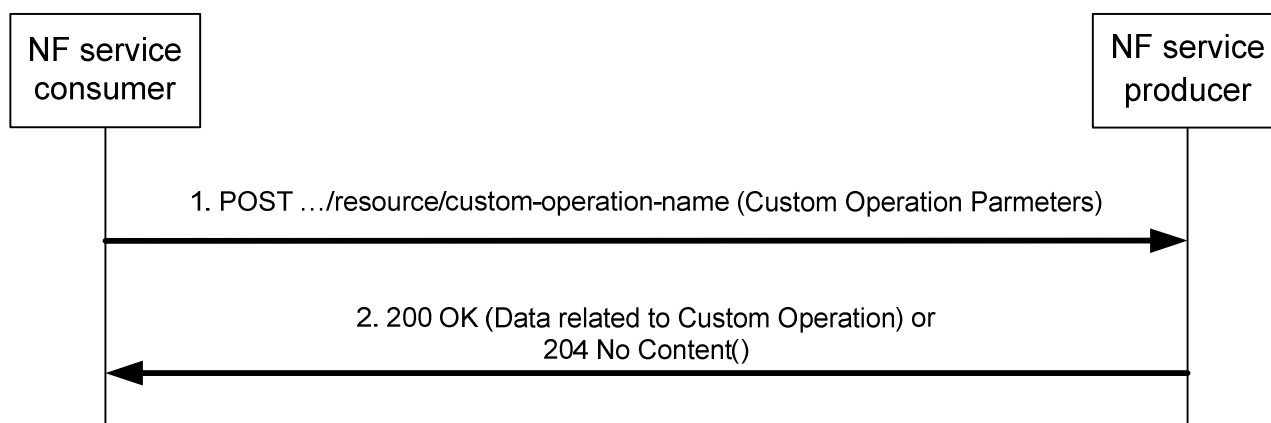
Figure 4.6.1.2-1: Custom Operation on a Resource using HTTP POST

1. The request URI identifies the custom operation to be executed and the resource the custom operation relates to and is constructed by adding a verb as name for the custom operation at the end of the resource URI (see clauses 4.4.2 and 5.1.3.2). Parameters for the custom operation are included in the request body.

2. On success, "204 No Content" or "200 OK" shall be returned. "200 OK" shall contain a body with data related to the custom operation.

On failure, the appropriate HTTP status code indicating the error shall be returned and appropriate additional error information should be returned in the POST response body (see clause 4.8).

Figure 4.6.1.2-2 illustrates the use of a custom operation related to a service.
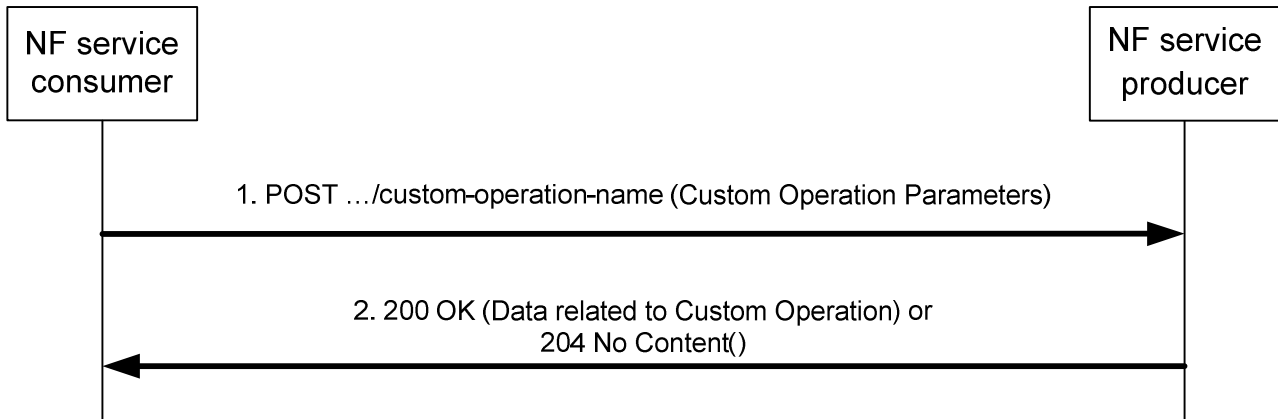
Figure 4.6.1.2-2: Custom Operation related to Service using HTTP POST

1. The request URI identifies the custom operation to be executed and is constructed by adding a verb as name for the custom operation at the end of the service URI (see clauses 4.4.2 and 5.1.3.2). Parameters for the custom operation are included in the request body.

2. On success, "204 No Content" or "200 OK" shall be returned. "200 OK" shall contain a body with data related to the custom operation.

On failure, the appropriate HTTP status code indicating the error shall be returned and appropriate additional error information should be returned in the POST response body (see clause 4.8).

4.6.1.3 Use of Asynchronous Operations

Certain service operations may be designed to allow the invocation of a request so that the response can be received asynchronously: if the NF service consumer when sending a request cannot expect to receive an immediate final response, the service consumer may provide a callback reference for final result notification. The service provider, when receiving a request that contains a callback reference for final result notification, may then return an immediate "202 Accepted", and notify the service consumer about the final result using the received callback reference at a later point in time.

4.6.1.4 Special provisions to support the seamless change of AMF as NF service producer

Services provided by the AMF can be transferred seamlessly to a new AMF when the corresponding UE context is transferred to that AMF.

To support a seamless change of the AMF as NF service producer, the procedures in clause 4.6.1 are applied with the following special provisions:

1. When becoming aware that a new AMF is serving the resource, the NF service consumer shall exchange the authority part of resource URIs with the address of a new NF service producer and shall use that URI in subsequent communication.

NOTE: An NF service consumer can become aware of an AMF change via Namf_Communication service AMFStatusChange Notifications, via Error response from old AMF, via link level failures (e.g. no response from the AMF), or via a notification from the NRF that the AMF has deregistered and can then determine the new AMF either via information received within those services or by selecting an AMF from an earlier received AMF set or the backup AMF.

2. Each AMF within a set of AMFs supporting seamless changes shall be prepared to receive updates for resource URIs constructed according to bullet 1 with the own IP address as authority part from the NF service consumer, by either handling the updates, or by replying with an HTTP "307 temporary redirect" error response pointing to new NF service producer, or by replying with another HTTP error such as an "404 Not found".

3. For a service that includes notifications from the AMF, the NF service consumer shall be prepared to receive notifications for the that service from any NF service producer within a set of NF service producers supporting seamless changes.

4.6.2 Use of Subscribe/Notify Communication

4.6.2.1 General

Subscribe/Notify communication between 5GC NFs can be used to keep involved NFs (consumers of a service) informed of data changes or events that occur at another NF (producer of the service). A notification is a message that contains information about the event.

Service consumer NFs (clients) need to subscribe to notifications at the service provider NF (server). This either happens explicitly by means of creating a new subscription resource (see clause 4.6.2.2), or implicitly by updating a relevant resource.

Service consumer NFs can in principle explicitly or implicitly subscribe to be notified about data change to any type of resource of any resource archetype (Document, Store or Collection). It is up to the API to define the resources that support subscriptions.

When the change/event occurs at the service producer NF, notifications (see clause 4.6.2.3) are sent from the service producer NF to the service consumer NFs. This communication initiated by the service producer to the service consumers requires that the service consumer NF (client) takes the role of an HTTP server and the service producer NF (server) takes the role of an HTTP client.

During the explicit subscription the service consumer NF (client) provides a callback URI and possibly additional filter criteria to the service producer NF (server). When the data-change/event occurs that matches the filter criteria in the subscription, the service producer NF (taking the role of an HTTP client) uses the provided callback URI to notify the service consumer NF (taking the role of an HTTP server) about the change.

4.6.2.2 Management of Subscriptions

4.6.2.2.1 General

The HTTP method to create a subscription shall be POST. The HTTP method to modify a subscription shall be PUT or PATCH. The HTTP method to delete a subscription (i.e. to unsubscribe) shall be DELETE (see IETF RFC 9110 [6]).

Subscriptions may be implicit, i.e. exist without being explicitly created by a dedicated subscribe operation.

Two types of implicit subscriptions exist:

1. The subscription is implied by an explicit operation different from the subscribe operation, which does not use the GET method. The subscription implied by the explicit operation and the corresponding notification shall be part of the same service.

2. The subscription exists without any explicit operation.

As an example for the first type, at the UDM the registered AMF (as long as it is registered) is implicitly subscribed to notification about de-registration and (possibly) P-CSCF restoration as side effect of the registration.

As another example for the first type, at the SMF, the AMF that created a SM Context for a PDU session is implicitly subscribed for SM Context Status notification. At AMF change the new AMF updates the SMF with its callback URI for receiving subsequent SM Context Status notification.

As an example for the second type, at the UDR any available UDM is implicitly subscribed to notification about changes of provisioned subscriber data. When provisioned subscriber data are modified at the UDR by means of provisioning, the UDR selects one of the available UDMs (i.e. one of the implicitly subscribed UDMs) and notifies it about the subscriber data change.

In the OpenAPI specification file, notifications for the second type of implicit subscriptions shall be specified as part of an explicit subscription.

4.6.2.2.2 Creation of a Subscription

Figure 4.6.2.2.2-1 illustrates explicit creation of a subscription.
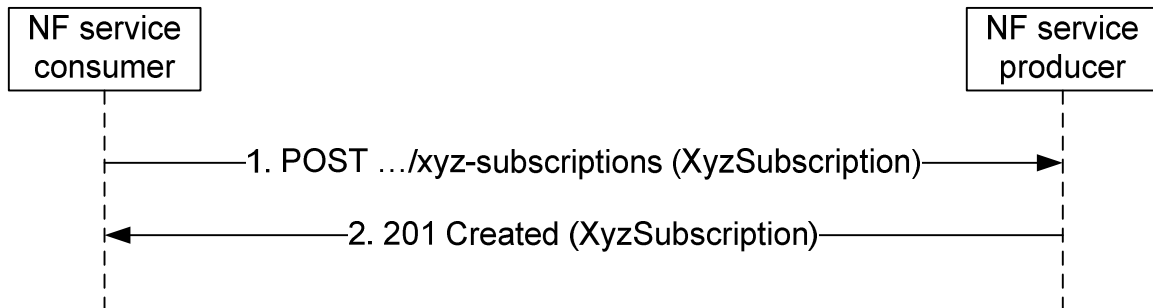
Figure 4.6.2.2.2-1: Creation of a subscription

The parent resource (collection of subscriptions) is identified by the request URI.

The data structure in the content of the POST request shall contain a callback URI, and may contain additional criteria to filter the set of events that trigger a notification. The request may contain an expiry time, suggested by the NF Service Consumer as a hint, representing the time up to which the subscription is desired to be kept active and the time after which the subscribed event shall stop generating notifications.

On success, "201 Created" shall be returned, the content of the POST response shall contain a representation of the created subscription, and the "Location" header shall contain the URI of the created resource. The created resource shall be served by the same NF (service) instance that received the service request, unless the 5GC SBI API specifications explicitly specified that in specific use cases the created resource may be served by another NF (service) instance. If in such specific use cases the resource is created in a different NF (service) instance, the identifier of the serving NF (service) instance shall be included in the response message.

If the HTTP scheme used in the returned URI is "https", then the authority of the URI included in the Location header shall be an FQDN, and not an IP address.

The response based on operator policies and taking into account the expiry time included in the request, may contain an expiry time (i.e. a future timestamp), as determined by the NF Service Producer, after which the subscription becomes invalid. If an expiry time was included in the request, then the expiry time returned in the response should be less than or equal to that value. Once the subscription expires, if the NF Service Consumer wants to keep receiving notifications, it shall create a new subscription in the NF Service Producer. The NF Service Producer shall not provide the same expiry time (i.e. a future timestamp) for many subscriptions in order to avoid all of them expiring and recreating the subscription at the same time. If the expiry time is not included in the response, the NF Service Consumer shall consider the subscription to be valid without an expiry time.

On failure, the appropriate HTTP status code indicating the error shall be returned and appropriate additional error information should be returned in the POST response body (see clause 4.9).

4.6.2.2.3 Modify a subscription

4.6.2.2.3.1 Modification of a Subscription Using HTTP PUT

Procedures that allow a NF service consumer to update the subscription at the server by means of a complete replacement shall use the HTTP PUT method to replace the current subscription with a new representation.

Figure 4.6.2.2.3.1-1 illustrates modification a subscription using HTTP PUT.
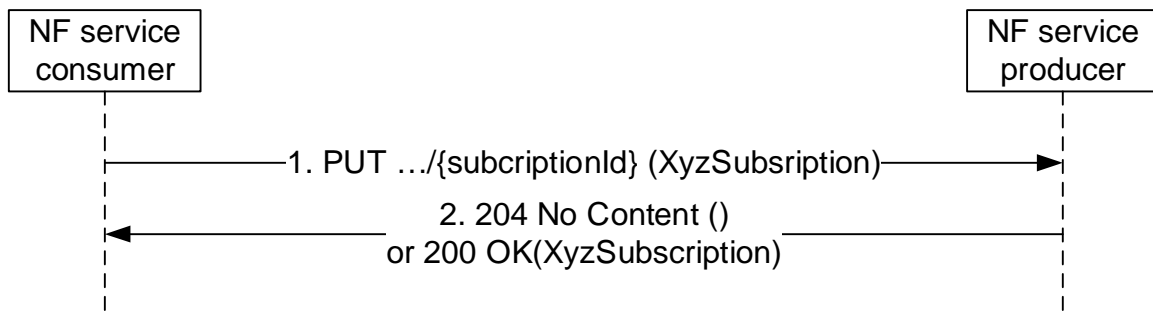
Figure 4.6.2.2.3.1-1: Modification a subscription using HTTP PUT

1. The NF Service Consumer shall send a PUT request to the resource URI representing the individual subscription. The content of the PUT request shall contain the subscription information to be replaced including the criteria to filter the set of events that trigger a notification. The request may contain an updated expiry time, suggested by the NF Service Consumer as a hint, to extend the subscription lifetime, representing the time upto which the subscription is desired to be kept active and the time after which the subscribed event shall stop generating notifications. If the request does not contain an expiry time, the NF Service Producer shall consider that the NF Service Consumer requests for an extension of the existing subscription lifetime without indicating any specific expiration time; still, the NF Service Producer shall be authoritative to set the expiry time in the subscription response according to its own policies.

2. On success, "204 No Content" without any response body or "200 OK" with a response body providing current resource representation shall be returned.

   When "200 OK" is returned, the response based on operator policies and taking into account the expiry time included in the request, may contain an expiry time (i.e a future timestamp), as determined by the NF Service Producer, after which the subscription becomes invalid. If an expiry time was included in the request, then the expiry time returned in the response should be less than or equal to that value. Once the subscription expires, if the NF Service Consumer wants to keep receiving notifications, it shall create a new subscription in the NF Service Producer, as specified in clause 4.6.2.2.2. The NF Service Producer shall not provide the same expiry time (i.e. a future timestamp) for many subscriptions in order to avoid all of them expiring and recreating the subscription at the same time. If the expiry time is not included in the response, the NF Service Consumer shall consider the subscription to be valid without an expiry time.

   When "204 No Content" is returned, it shall be interpreted that the NF Service Producer accepted entirely the resource representation provided by the NF Service Consumer in the request; e.g., if the request contained a proposed expiry time, a 204 response shall be interpreted as if such timestamp is accepted by the NF Service Producer as the expiration time for the subscription and, similarly, if the request did not contain a proposed expiry time, a 204 response shall be interpreted as if no expiration time is set by the NF Service Producer for the subscription.

On failure, the appropriate HTTP status code indicating the error shall be returned and appropriate additional error information should be returned in the PUT response body (see clause 4.8).

If the NF Service Consumer is not allowed to update the subscription information, the "403 Forbidden" HTTP status code shall be returned and appropriate additional error information should be returned in the PUT response body (see clause 4.8).

If the resource that is to be updated does not exist at the NF service producer, the "404 Not Found" HTTP status code shall be returned.

4.6.2.2.3.2 Modification of a Subscription Using HTTP PATCH

Procedures that allow a NF service consumer to update subscription at the server by means of a partial replacement shall use the HTTP PATCH method (see IETF RFC 5789 [10]) to modify the current subscription according to given modification instructions.

Figure 4.6.2.2.3.2-1 illustrates updating a resource using HTTP PATCH.
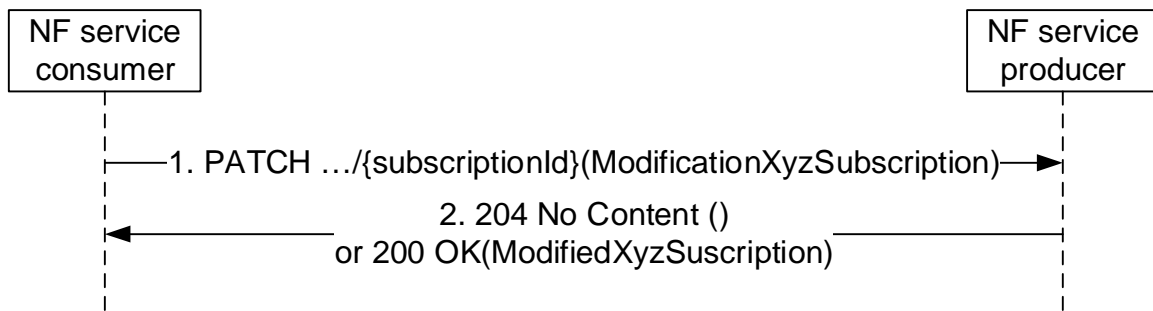
Figure 4.6.2.2.3.2-1: Modification a subscription using HTTP PATCH

1. The NF Service Consumer shall send a PATCH request to the resource URI representing the individual subscription. The content of the PATCH request shall contain the modification instructions. The request may contain an expiry time (i.e. a future timestamp), requested by the NF Service Consumer, representing the time upto which the subscription is desired to be kept active and the time after which the subscribed event shall stop generating notifications.

2. On success, "204 No Content" without any response body or "200 OK" with a response body containing the modified subscription information shall be returned. When "204 No Content" is returned and if the request included an expiry time, then the requested expiry time shall be accepted by the NF Service Producer. When "200 OK" is returned and if the request included an expiry time then the response based on operator policies and taking into account the expiry time included in the request, shall contain an expiry time (i.e. a future timestamp), as determined by the NF Service Producer, after which the subscription becomes invalid. If an expiry time was included in the request, then the expiry time returned in the response should be less than or equal to that value. Once the subscription expires, if the NF Service Consumer wants to keep receiving notifications, it shall create a new subscription in the NF Service Producer, as specified in clause 4.6.2.2.2. The NF Service Producer shall not provide the same expiry time (i.e. a future timestamp) for many subscriptions in order to avoid all of them expiring and recreating the subscription at the same time.

On failure, the appropriate HTTP status code indicating the error shall be returned and appropriate additional error information should be returned in the PATCH response body (see clause 4.8).

4.6.2.2.4 Delete a subscription

Figure 4.6.2.2.4-1 illustrates explicit deletion of a subscription.
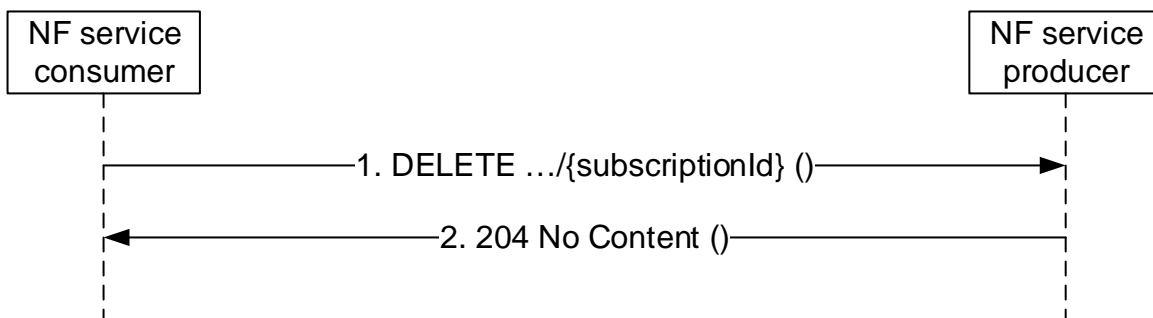
Figure 4.6.2.2.4-1: Deletion of a subscription

1. The NF Service Consumer shall send a DELETE request to the resource URI representing the individual subscription. The request body shall be empty.

2. On success, "204 No Content" shall be returned. The response body shall be empty.

On failure, the appropriate HTTP status code indicating the error shall be returned in the DELETE response body (see clause 4.8).

4.6.2.3 Notifications

The HTTP method for the notification that corresponds to an explicit subscription shall be POST (see IETF RFC 9110 [6]).

NOTE: Clause 5.3.7 describes how to encode Notifications in OpenAPI specification files.

Figure 4.6.2.3-1 illustrates a notification.
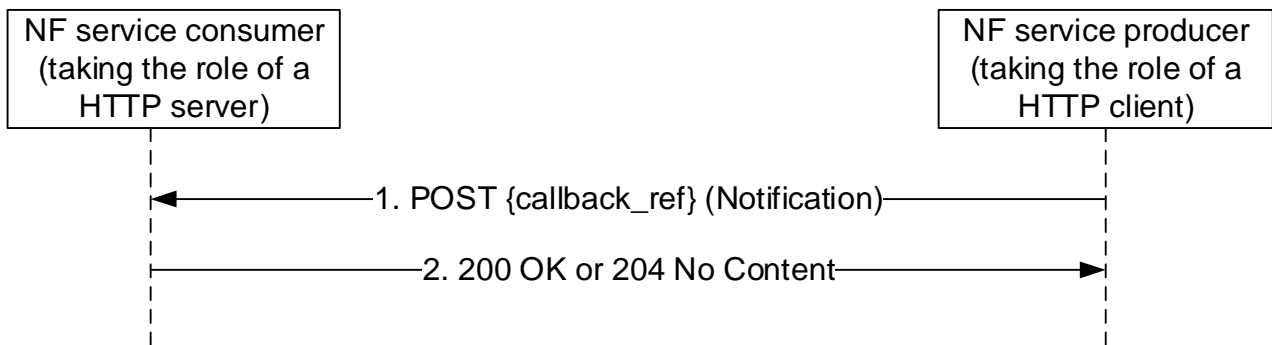
Figure 4.6.2.3-1: Notification

1.   The callback reference provided during creation of the subscription resource, or otherwise known from implicit subscription, is used as the request URI. The callback reference for implicit subscriptions are obtained from the NRF. When an NF / NF service registers with the NRF, the default notification subscriptions along with the callback URI for receiving those notifications may be provided (see clause 6.1.6.2.3 of 3GPP TS 29.510 [18]).

The content of the POST request shall contain the notification content.

The content of the notification should follow the resource definition of the subscribed resource and can for example be based on the resource definition of the GET operation, but it is up to the API to define the notification resource definition.

Each API that supports subscription to collection/store archetype resources, should specify in their semantics whether notifications should be sent by changes on the collection/store resource ONLY (i.e. creation/deletion of the main top-level resource itself, and creation/deletion of its children), or if in addition the consumer can expect to get notifications from changes on the resource representation.

2.   On success, "200 OK" shall be returned if any information needs to be included in the content of the POST response; otherwise, "204 No Content" shall be returned and the content of the POST response shall be empty.

On failure, the appropriate HTTP status code indicating the error shall be returned and appropriate additional error information should be returned in the POST response body (see clause 4.8).

4.6.2.4 Special provisions to support the seamless change of AMF as NF service consumer

Services consumed by an AMF can be transferred seamlessly to a new AMF when the corresponding UE context is transferred to that AMF.

To support a seamless change of AMF as NF service consumer, the procedures in clause 4.6.2 are applied with the following special provisions:

1.   When becoming aware that a new AMF is requiring notifications related to a subscription resource, the NF service producer shall exchange the authority part of the corresponding Notification URI with the address of that new NF service consumer and shall use that URI in subsequent communication.

NOTE: An NF service producer can become aware of an AMF change via Namf_Communication service AMFStatusChange Notifications, via Error response from old AMF, via link level failures (e.g. no response from the AMF), or via a notification from the NRF that the AMF has deregistered and can then determine the new AMF either via information received within those services or selecting an AMF from an earlier received AMF set or the backup AMF.

2. Each AMF within a set of AMFs supporting seamless changes shall be prepared to receive notifications at the Notification URI constructed according to bullet 1 with the own IP address as authority part from the NF service producer, by either handling the notifications, or by replying with an HTTP "307 temporary redirect" error response pointing to new NF service consumer, or by replying with another HTTP error such as an "404 Not found".

4.7 HATEOAS

4.7.1 General

As defined in [14], HATEOAS stands for Hypermedia As The Engine Of Application State. It means that the hypermedia models application state transitions and describe application protocols.

As defined in [15] clause 3 RESTful Domain Application Protocols, an application is a software implementation defined to achieve a particular goal. It consists of a set of constrained interactions between NF Service Consumer and Producer performed at run-time that are guided by an application specific set of rules. The application transits across some intermediate states until the application's goal is achieved. The application has then reached its final state.

An application state is a snapshot of an application instance.

On each interaction, the NF Service Consumer and Producer exchange representations of resource state. According to [14], "REST concentrates all of the control state into the representations received in response to interactions." and "The model application is therefore an engine that moves from one state to the next by examining and choosing from among the alternative state transitions in the current set of representations." After each interaction the NF Service Consumer is then presented with control state options to interact with additional resources. These control states are in the form of hypermedia markups embedded in the returned resource representation. The application state changes when an NF Service Consumer examines and chooses which control to operate and subsequently interacts with the resources identified in the selected control state.

HATEOAS support is optional. If HATEOAS is supported, the procedure in the present clause 4.7 shall apply.

4.7.2 3GPP hypermedia format

4.7.2.1 Basic 3GPP hypermedia format

NOTE 1: Basic 3GPP hypermedia format is derived from Hypertext Application Language (HAL). HAL is specified in an expired internet draft available at "https://tools.ietf.org/html/draft-kelly-json-hal-08".

Basic 3GPP hypermedia format specifies the following optional reserved properties (see 3GPP TS 29.571 [5] for the complete list and definition of objects and object members):

- "_links": contains links to other resources and expresses valid state transitions.

A NF service producer shall construct a basic 3GPP hypermedia document by taking a 3GPP defined JSON object attribute list and then adding a "_links" attribute.

Table 4.7.2.1-1: _links attribute

Attribute name	Data type	P	Cardinality	Description
_links	map(LinksValueSchema)	C	0..N	_links attribute to be added into the JSON hypermedia object definition

The LinksValueSchema data type shall be added to the list of re-used data types of the hypermedia enabled API (see 3GPP TS 29.509 [21] for an example of implementation of a hypermedia API).

NOTE 2: Depending of the applicable situation, the presence condition and the cardinality can be changed in accordance. LinksValueSchema data type is defined in 3GPP TS 29.571 [5].

The "_links" member names are link relation types (as defined by IETF RFC 8288 [11]) and values are either a "link" object or an array of "link" objects.

Basic 3GPP hypermedia format specifies the following "link" attribute:

- "href": contains the URI of the linked resource.

A NF service producer shall set the Content-Type HTTP header to "application/3gppHal+json" when returning an HTTP contentwith a basic 3GPP hypermedia enabled document.

A NF service consumer supporting HATEOAS shall advertise it by adding an "Accept" HTTP header with "application/3gppHal+json" as media type.

NOTE 3: The HATEOAS principle relies on NF Service Producer providing control state options (_links objects) embedded in the returned resource representation to the NF Service Consumer. An NF Service Consumer may decide to use the format of the _links attribute in HTTP requests to transfer URIs. This is beyond the scope of HATEOAS and another content type than "3gppHal+json" such as "application/json" can be used.

4.7.2.2 Extended 3GPP hypermedia format

NOTE 1: Extended 3GPP hypermedia format is derived from HAL-FORMS media type, see "https://rwcbook.github.io/hal-forms/".

Extended 3GPP hypermedia format specifies the following optional reserved properties (see 3GPP TS 29.571 [5] for the complete list and definition of objects and object members):

- "_links": contains links to other resources and expresses valid state transitions;

- "_templates": contains the HAL templates corresponding to the _links attribute.

A NF service producer shall construct a extended 3GPP hypermedia document by taking a 3GPP defined JSON object attribute list and then adding "_links" attribute and "_templates" attribute.

Table 4.7.2.2-1: _halForms attribute

Attribute name	Data type	P	Cardinality	Description
_links	map(Link)	C	1..N	The key is the name of the link relation type.
_templates	map(HalTemplate)	C	1..N	The key is the name of the corresponding link relation type to which the template is to be applied.

The Link data type and HalTemplate data type shall be added to the list of re-used data types of the hypermedia enabled API.

NOTE 2: Depending of the applicable situation, the presence condition can be changed in accordance. Link and HalTemplate data types are defined in 3GPP TS 29.571 [5].A NF service producer shall set the Content-Type HTTP header to "application/3gppHalForms+json" when returning an HTTP contentwith Extended 3GPP hypermedia enabled document.

A NF service consumer supporting HATEOAS and extended 3GPP hypermedia shall advertise it by adding an "Accept" HTTP header with "application/3gppHalForms+json" as media type.

4.7.3 Advertising legitimate application state transitions

When a NF service producer responds to a NF service consumer and there is one or more application state transition possible, the NF service producer shall advertise them by adding a "_links" property in the returned resource representation. When there are multiple state transitions with different relation types, then one member per relation type shall be added to the "_links" object which name is equal to the relation type. If there is only one state transition for a given relation type then the value of the member is a "link" object otherwise it is an array of "link" objects.

A NF service producer shall include a link into the returned resource representation with a registered relation type "self" when it is expected further actions upon it (for instance reading it again or replacing the resource state).

NOTE 1: For a hypermedia application, a returned representation without any link denotes for the NF service consumers the end of the interaction with the NF service producer. 3GPP APIs does not fulfil this rule.

4.7.4 Inferring link relation semantic

When a NF service consumer receives a response with linked resources then it shall infer the link relation semantic from the relation type. It shall not infer it from the linked resource URI format.

In 3GPP hypermedia, relation types are the name of "_links" object members.

4.7.5 Common Relation Types

4.7.5.1 Introduction

This clause contains the list of relation types supported in 3GPP Service Based Interface APIs.

As defined in IETF RFC 8288 [11] clause 2.1, a link relation type identifies the semantics of a link. It describes how resources are related to each other. It may also be used to indicate that the target resource of a link has particular attributes, or exhibits particular behaviours. Relation types shall not be confused with media types. It does not identify the format of the representation that results when the link is dereferenced.

There are two kinds of relation types:

- Registered relation types;

- Extension relation types.

Registered relation types are identified by a token (for instance "self") and can be reused by other applications such as 3GPP SBI APIs. They are registered by IANA. Registered relation types shall be preferred against extension relation types when expressing the link relation between two resources.

If there is a need to define a relation type that does not correspond to a registered one but it is not wanted to register it then an extension relation type shall be used instead.

4.7.5.2 Registered relation types

The "Link Relations" registry is located at: https://www.iana.org/assignments/link-relations.

Table 4.7.5.2-1 specifies the list of registered relation types supported by all hypermedia enabled 3GPP APIs.

Table 4.7.5.2-2 specifies the list of registered relation types that can be used by some hypermedia enabled 3GPP APIs, depending on the API design.

Table 4.7.5.2-1: mandatory registered relation types

Relation name
self

Table 4.7.5.2-2: optional registered relation types

Relation name
next
first
previous
last
item

4.7.5.3 Extension relation types

When no registered relation exists to express the relation between two resources, an extension relation type shall be used instead. It may be defined as a string token or as a URI as defined in IETF RFC 8288 [11].

An API specification using extension relation types shall contain a clause "Relation types" in the clause "Simple data types and enumerations" (see 3GPP TS 29.509 [21] for an example of implementation of a hypermedia API). The clause shall contain a table listing the token or the URI of the created relation types. It shall also contain a detailed

specification of the semantic of the relation types defining the conditions that the NF Service Consumer shall match to follow a link.

4.7.6 Negotiating the support of optional HATEOAS features

The supported feature mechanism in clause 6.6.2 of 3GPP TS 29.500 [2] should be used to negotiate the usage of optional HATEOAS features in addition to negotiating the content type "3gppHal+json" and "3gppHalForms+json". Separate supported features can be defined for link relation types related to different use cases.

4.8 Error Responses

4.8.1 Error Response Status Code

When an error occurs that prevents the NF/NF service acting as an HTTP server from successfully fulfilling the HTTP request, the NF/NF service shall map an application error to the most similar 4xx/5xx HTTP status code as defined in clause 5.2.7 of 3GPP TS 29.500 [2].

4.8.2 Error Response Body

When the HTTP status code is not enough for the NF/NF service acting as an HTTP client to determine the cause of the error, the NF/NF service acting as an HTTP server should provide additional application related error information, by including in the response body a representation of a "ProblemDetails" data structure according to IETF RFC 9457 [19] that provides additional details of the error.

NOTE 1: The response body with the "ProblemDetails" data structure does not need to be sent on a 3GPP 5GC API for a particular HTTP status code if that HTTP status code itself provides enough information of the error, or if there are security concerns disclosing detailed error information.

The definition of the general "ProblemDetails" data structure from IETF RFC 9457 [19] is specified in clause 5.2.4.1 of 3GPP TS 29.571 [5]. The "ProblemDetails" data structure is a JSON object, as defined in IETF RFC 9457 [19], and contains the following attributes:

a) "type" - a URI reference according to IETF RFC 3986 [9] that identifies the problem type;

b) "title" - a short, human-readable summary of the problem type that should not change from occurrence to occurrence of the problem;

c) "status" - the HTTP status code for this occurrence of the problem;

d) "detail" - a human-readable explanation specific to this occurrence of the problem; and

e) "instance" - a URI reference that identifies the specific occurrence of the problem.

A particular API may define additional attributes that provide more information about the error.

NOTE 2: IETF RFC 9457 [19] allows adding of new properties in the "ProblemDetails" object.

The following additional attributes are generic extensions defined for the 3GPP 5GC APIs:

a) "cause"- a machine-readable application error cause specific to this occurrence of the problem; and

b) "invalidParams" - invalid parameters causing a request to be rejected.

The "cause" attribute should be included and provide application-related error information, if available. Application error causes should be defined in 5GC SBI APIs specifications, using the UPPER_WITH_UNDERSCORE case convention specified in clause 5.1.1.

EXAMPLE 1: "OUT_OF_LADN_SA".

The "invalidParams" attribute should be used to report invalid parameters when a request is rejected due to invalid parameters.

All the application error causes supported by an API should be defined in a specific clause "Application Errors" under the "Error Handling" clause specified for the API. The application error causes that a specific service operation may respond should be further listed in the table defining the data structure supported by the response body, with the associated HTTP error status code.

To enable a SEPP or an SCP for Indirect Communications to provide error details in error responses they originate, all service operations should support returning error responses including a representation of a "ProblemDetails" data structure. If additional application specific information is required in the error responses, the API should support returning the additional applicative information as specified in clause 4.8.3. The NF/NF service that generates the HTTP response shall include in the HTTP response a "Content-Type" header field set to:

- "application/problem+json", if the response includes a content containing the "ProblemDetails" (or extended ProblemDetails, see clause 4.8.3) data structure; or

- "application/json", if the response includes a content containing an application-specific data structure.

    NOTE 3: Designing an error response content containing an application-specific data structure is discouraged for new APIs or service operations as this prevents a SEPP or an SCP when using indirect communications to include error details for errors they originate. If an API is defined with an application-specific error response body, the API also needs to support an error response body encoding a ProblemDetails object.

4.8.3 Extending ProblemDetails for API Backward Compatibility

For a service operation that returns "ProblemDetails" in error responses in a given release, if in a later release it is required to provide additional application specific information in the error responses, the API should be modified to return an Extended-ProblemDetails data type by reusing the "ProblemDetails" common data type, as specified in clause 5.3.17, to keep the API backward compatibility.

The "Content-Type" header shall be set to "application/problem+json" for the error response with the content containing the Extended-ProblemDetails data type define above.

    EXAMPLE:

- an "AdditionInfo<ServiceOperation>" structured data type containing the additional information to be returned, as specified in clause 5.2.4.2:

Table 4.8.3-1 Definition of type AdditionInfo<ServiceOperation>

Attribute name	Data type	P	Cardinality	Description	Applicability
<attributes of additional information>	"*<type>*" or "array*(<type>)*" or "map*(<type>)*" or "Any Type"	"M", "C" or "O"	"0..1", "1" or "M..N"	<only if applicable>	

- a "ProblemDetails<ServiceOperation>" data type as combination of "ProblemDetails" common data type and "AdditionInfo<ServiceOperation>" data type define above, as specified in clause 5.2.4.5:

Table 4.8.3-2 Definition of type ProblemDetails<ServiceOperation> as a list of to be combined data types

Data type	Cardinality	Description	Applicability
ProblemDetails	1	<only if applicable>	
AdditionInfo<ServiceOperation>	1	<only if applicable>	

Example OpenAPI specification of ProblemDetails<ServiceOperation> data type, as specified in clause 5.3.10:

```
components:
  schemas:
    ProblemDetails<ServiceOperation>:
      allOf:
      - $ref: 'TS29571_CommonData.yaml#/components/schemas/ProblemDetails'
```

```
      - $ref: '#/components/schemas/AdditionInfo<ServiceOperation>'
```

Example OpenAPI specification of an error response returning ProblemDetails<ServiceOperation> data type:

```
responses:
  '504':
    description: Gateway Timeout
    content:
      application/problem+json:
        schema:
          $ref: '#/components/schemas/ProblemDetails<ServiceOperation>'
```

4.9 Transferring multiple resources to a NF Service Consumer

4.9.1 General

This clause describes some possible options that an API may implement when a NF Service Producer needs to return the representations of multiple resources to a NF Service Consumer, e.g. during the query of a large collection of resources (see clause 4.6.1.1.2.2).

Which options an API may support is defined in the respective stage 3 specification of the API.

4.9.2 Direct Delivery

A NF Service Producer may return the representations of the resources directly in the response body, i.e. the response body contains an array of the resource representations.

4.9.3 Direct Delivery with Iterations

If a large number of resource representations need to be returned, the NF Service Producer may return a representation containing a partial list of the requested resources in the response body, with link(s) containing URI(s) allowing the client to retrieve the remaining part(s) of the resources.

The returned representation containing a partial list of the requested resource is a "3gppHal+json" document. The document is a JSON object with two members whose names are below.

- _links.

- child: contains the resources of the partial list.

The member whose name is "_links" shall contain a member whose name is "self" and whose value is a "link" object that contains the URI of the returned representation. It shall also contain a member whose name is "next" and whose value is a "link" object that contains the URI of the next partial list of the collection if the returned partial list is not the last one.

The member whose name is "_links" should also contain members whose names are "first", "previous" and "last" and whose values contain a "link" object that contains the URIs of the first, previous and last partial lists of the collection if such lists exist.

The returned representation shall have a member whose name is "child" and whose value is an array of objects. Each of the individual resource representations returned in the partial list shall be embedded in an object of that array. Each object shall also have a member whose name is "_links". The later shall contain a member whose name is "self" and whose value is a "link" object that contains the URI of the embedded representation.

The table below provides a template to be added in the clause describing the GET operation of a 3GPP API using the direct delivery with iteration mechanism.

Table 4.9.3-1: Data structures supported by the GET Response Body on this resource

Data type	P	Cardinality	Response codes	Description
PartialList	M	1	200 OK	This case represents a successful return of a partial list for the corresponding request with direct delivery with iteration.

The following data types shall be added to the list of specific data types and described as below in the structured data type clause.

Table 4.7.2-1: PartialList

Attribute name	Data type	P	Cardinality	Description
_links	map(LinksValueSchema)	M	1..N	contains the pagination links
child	array(ApiSpecificHypermediaEnabledIndividualResource)	M	1..N	contains the individual resources with a self link. The data type in the array is specific to the API and is a hypermedia enabled version of the individual resource data type.

Table 4.7.2-1: ApiSpecificHypermediaEnabledIndividualResource

Attribute name	Data type	P	Cardinality	Description
attribute1				
attribute2				
…				
attribute N				
_links	selfLink	M	1	contains the link to itself

NOTE 1: attributes 1 to N are the attributes of the original individual resource.

The LinksValueSchema and SelfLink data types shall be added to the list of re-used data types of the 3GPP API.

A NF Service Consumer that receives link(s) in the response body may retrieve the remaining part(s) of the resources by sending GET requests towards the URI(s) contained in the link(s).

4.9.4   Indirect Delivery

A NF Service Producer may not return any requested resource representation and instead may return a representation containing only a list of links to the requested resources in the response body.

The returned representation containing the list is a "3gppHal+json" document. The document is a JSON object with one member whose name is:

-    _links.

This member shall contain a member whose name is "item" and whose value is an array of "link" objects. Each of the link objects contains one requested resource URI. There shall be one link object per requested resource.

It shall also contain a member whose name is "self" and whose value is a "link" object that contains the URI of the returned representation.

A NF Service Consumer that receives such a response may then send a GET request per resource URI to retrieve the requested resources from the NF Service Producer.

4.9.5   Indirect Delivery with HTTP/2 Server Push

A NF Service Producer may use HTTP/2 Server Push, if HTTP/2 Server Push is supported in the PLMN.

To use HTTP/2 Server Push, the NF Service Producer shall send PUSH_PROMISE frames in the HTTP response, with each PUSH_PROMISE frame containing a GET request targeting the URI of one resource to be transferred and the

reserved stream identifier to be used for transferring the resource. Then the NF Service Producer shall send Push Responses via the corresponding reserved streams, with each Push Response containing the representation of the associated resource. The NF Service Producer shall also send links with the URIs of the resources in DATA frame(s) of the response message.

A NF Service Consumer may disable HTTP/2 Server Push by sending SETTINGS_ENABLE_PUSH parameter with value "0" on HTTP level, as specified in IETF RFC 9113 [13].
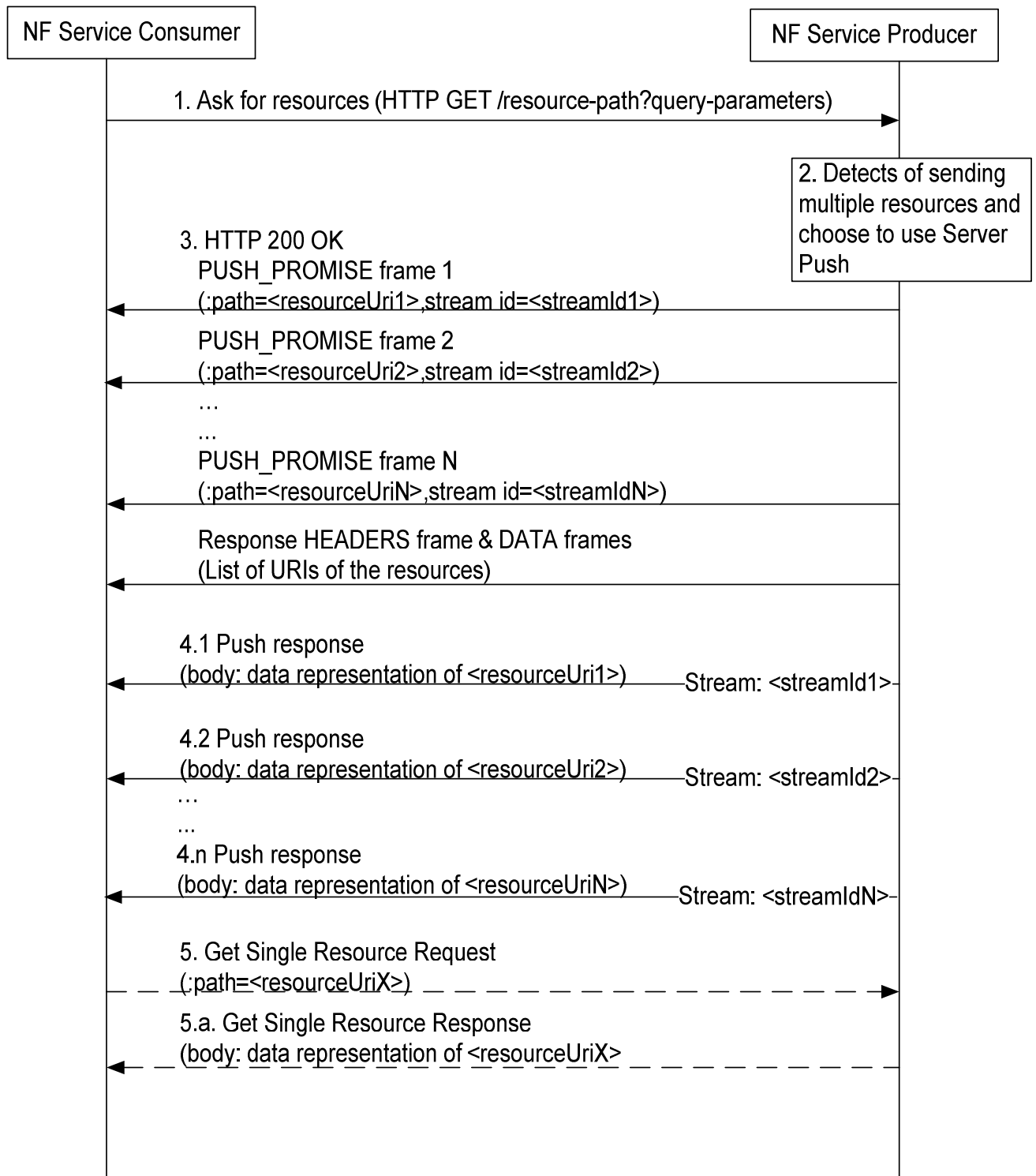
Figure 4.9.5-1 Indirect Delivery with HTTP/2 Server Push

1. A NF Service Consumer sends a HTTP request to get resources(s) to the NF Service Producer, e.g. a query of a collection of resources.

2. The NF Service Producer detects that multiple resources are to be returned and choose to indirectly deliver the resources with the Server Push mechanism.

3. The NF Service Producer returns multiple PUSH_PROMISE Requests before HEADERS frame and DATA frames(s) to the NF Service Consumer. Each PUSH_PROMISE Request contains the URI of one resource to be transferred and the identifier of the reserved stream used for transferring the resource. The NF Service Producer shall also send links with the URIs of the resources in DATA frame(s) of the response message.

4.1-4.n. The NF Service Producer sends Push Reponses via corresponding reserved streams. Each Push Response contains the representation of the associated resource.

5. If the NF Service Consumer does not successfully receive a resource in time, it may send a request to get that resource, using the resource URI previously received from the Push Request.

5.a. The NF Service Producer returns the data of the requested resource in the response.

4.9.6   Criteria for choosing the transfer method

The following considerations may be used to determine which method to use transfer multiple resources to a NF Service Consumer.

If the size of the representation of each resource is small, direct delivery is preferred. If the number of resources to be returned is large, the NF Service Producer may choose iterative delivery.

NOTE 1: For this release of this specification, a JSON content size less than 64000 octets is considered as not large and a JSON content size larger than 64000 octets is considered as large.

If the size of the representation of each resource is large, indirect delivery is preferred. If the NF Service Producer supports HTTP/2 Server Push, then:

-   when SETTINGS_ENABLE_PUSH parameter with value "1" has been received from the NF Service Consumer, as specified in IETF RFC 9113 [13], it should choose HTTP/2 Server Push to deliver the resource.

-   when SETTINGS_ENABLE_PUSH parameter with value "0" has been received from the NF Service Consumer, as specified in IETF RFC 9113 [13], it must not choose HTTP/2 Server Push to deliver the resources.

-   when SETTINGS_ENABLE_PUSH parameter has not been received from the NF Service Consumer, as specified in IETF RFC 9113 [13], it may decide whether to use HTTP/2 Server push or not, depending on other factors, e.g. operator policy, whether the client and server pertain to the same PLMN, etc.

An NF Service Producer shall use Indirect Delivery with HTTP/2 Server Push only if the NF Service Consumer (client) indicated support for accepting server pushed resource representations, via the Supported Features negotiation as specified in clause 6.6.2 of 3GPP TS 29.500 [2].

NOTE 2: In this release the Indirect Delivery with HTTP/2 Server Push is not used by 3GPP service based interface APIs.

4.10   Scopes definition for OAuth2.0 access token

As indicated in 3GPP TS 33.501 [22] and in clause 6.7.3 of 3GPP TS 29.500 [2], the access to an 5GC API may be authorized by means of the OAuth 2.0 protocol (see IETF RFC 6749 [23]), based on local configuration. OAuth 2.0 supports the concept of scope values to signal which actual access rights an access token represents.

Each 5GC API shall define a service-level scope set to the service name of the NF Service. This scope grants generic access to the given API, for those operations on resources that do not require a specific authorization. In addition, a 5GC API may define additional resource/operation-level scopes, that uniquely represents the type of operation (e.g. create/modify/read), the resource and the service; a resource/operation-level scope and the service-level scope, together, grant access to the operations on resources that require a specific authorization.

The need for defining additional resource/operation-level scopes in a 5GC API depends on the API functionalities and service operations and on the expected NF service consumers, and thus is up to the API responsible Working Group to decide. The following points may be considered when assessing the need for additional scopes:

- It should be carefully assessed during the design of the 5GC API whether different access rights to the functionalities provided by the API may be needed for different NF service consumers, e.g. considering example NF service consumers of the different service operations supported by the API.

- A 5GC API defined with a service-level scope only, i.e. without additional resource/operation-level scopes, grants access permissions to all its service operations to all NF service consumers allowed to access the service. It should be ensured that designing the API in this way does not cause any vulnerability to the 5GS, e.g. by granting inappropriate read access to sensitive data to undue NF service consumers or by granting inappropriate write (or delete) access to undue NF service consumers.

- Additional scopes should be defined when it is needed to enable restricting the access to certain resource/operations of the 5GC API to specific NF service consumers. This may be the case e.g. when an API supports multiple service operations that may be invoked by different NF service consumers (possibly with different NF types or belonging to different S-NSSAIs, PLMNs or SNPNs), or when it is needed to differentiate access rights for read vs. create/modify/write operations.

- The definition of additional scopes comes with some implementation and network operation complexity. Care should be taken not to increase unreasonably the number of scopes defined in a 5GC API.

- Each additional resource/operation-level scope shall be defined with a clear description (in the main body and in the OpenAPI file of the Technical Specification defining the API) of the access right the additional scope corresponds to. The naming of the resource/operation-level scopes should follow the conventions defined in clause 5.3.16 for ease of understanding and consistency across all 5GC APIs.

5 Documenting 5GC SBI APIs

5.1 Naming Conventions

5.1.1 Case Conventions

The following case conventions for names and strings are used in the 5GC SBI service APIs.

1) UPPER_WITH_UNDERSCORE

   All letters of a string are capital letters. Digits are allowed. Word boundaries are represented by the underscore "_" character. No other characters are allowed.

   Example 1:

   a) DATA_MANAGEMENT

   b) CELL_CHANGE

2) lower_with_underscore

   All letters of a string are lowercase letters. Digits are allowed. Word boundaries are represented by the underscore "_" character. No other characters are allowed.

   Example 2:

   a) data_management;

   b) cell_change.

3) UPPER-WITH-HYPHEN

   All letters of a string are capital letters. Digits are allowed. Word boundaries are represented by the hyphen "-" character. No other characters are allowed.

Example 3:

    a) DATA-MANAGEMENT

    b) CELL-CHANGE

4) lower-with-hyphen

All letters of a string are lowercase letters. Digits are allowed. Word boundaries are represented by the hyphen "-" character. No other characters are allowed.

Example 4:

    a) data-management;

    b) cell-change.

5) UpperCamel

A string is formed by concatenating words. Each word starts with a letter or a digit. The first letter of each word shall be an uppercase letter; all other characters in the word shall be lowercase letters or digits.

Abbreviations follow the same scheme (i.e. first letter uppercase, all other letters lowercase).

Example 5:

    a) DataManagement.

    b) CellChange

    c) 5QiPriorityLevel

    d) Amf3GppAccessRegistration

6) lowerCamel

A string is formed by concatenating words.Each word starts with a letter or a digit. The first letter of the first word shall be a lowercase letter; the first letter of the rest of the words shall be an uppercase letter. All other characters in the words shall be lowercase letters or digits.

Abbreviations follow the same scheme.

Example 6:

    a) dataManagement

    b) cellChange

    c) 5qiPriorityLevel

NOTE: These naming conventions are used as guidelines, to provide some uniformity in the specification of the different 5GC APIs. However, for different reasons, sometimes exceptions can be made. In any case, the OpenAPI specifications are mandatory, and the different network elements cannot determine that a certain message, that otherwise complies to the OpenAPI specification, is incorrect based only on the fact that it does not follow a given naming convention in a certain data element.

5.1.2 API Naming Conventions

An API shall take the name of the corresponding service (e.g. Nudm_SubscriberDataManagement). When used in URIs the name shall be converted to lower-with-hyphen and may use an abbreviated form (e.g. nudm-sdm).

5.1.3 Conventions for URI Parts

5.1.3.1 Introduction

The parts of the URI syntax that are relevant in the context of the 5GC SBI service APIs are as follows:

- Path, consisting of segments, separated by "/" (e.g. segment1/segment2/segment3).

- Query, consisting of pairs of parameter name and value (e.g., ?mcc=262&mnc=01, where two pairs are presented).

5.1.3.2 URI Path Segment Naming Conventions

a) All path segments of a resource URI which represent a string constant shall use lower-with-hyphen (this implies that a path cannot end with "/").

Example 1:

subscriber-data

b) If a resource represents a collection of entities and the last path segment of the resource URI is a string constant, that last path segment shall be plural.

Example 2:

…/prefix/api/v1/users

c) For resources where the last path segment of the resource URI is a string constant, that last path segment shall be a noun or a composite noun.

Example 3:

…/prefix/api/v1/users

Example 4:

…/prefix/api/v1/user-session

d) For custom operations, the last path segment of the URI via which the operation is invoked shall be a verb, or shall start with a verb.

Example 5:

…/app_instances/{appInstanceId}/instantiate

Example 6:

…/sessions/terminate-all

e) All path segments of a URI which are variable names shall use lowerCamel, and shall be surrounded by curly brackets.

Example 7:

…/subscriber-data/{supi}

f) Once a variable is replaced at runtime by an actual string, the string shall follow the rules for a path segment defined in IETF RFC 3986 [9]. IETF RFC 3986 [9] disallows certain characters from use in a path segment. Each actual 5GC SBI service API specification shall define this restriction to be followed when generating values for path segment variables, or propose a suitable encoding (such as percent-encoding according to IETF RFC 3986 [9]), to escape such characters if they can appear in input strings intended to be substituted for a path segment variable.

5.1.3.3 URI Query Naming Conventions

a) URI query parameter names in queries shall use lower-with-hyphen.

Example 1:

> ?nf-type=AMF

b) Variables that represent actual parameter values in queries shall use lowerCamel and shall be surrounded by curly brackets.

Example 2:

> ?nf-id={chooseAValue}

c) When a variable is replaced at runtime by an actual string, the convention defined in clause 5.1.3.2 item f) applies to that string.

5.1.4 Conventions for Names in Data Structures

The following syntax conventions apply when defining the names for attributes in the 5GC SBI service API data structures, carried in the content of http requests and responses.

a) Names of attributes shall be represented using lowerCamel.

Example 1:

> attributeName

b) Names of arrays (i.e. those with cardinality 1..N or 0..N) shall be plural rather than singular.

Example 2:

> users

c) Each value of an enumeration type shall be represented using UPPER_WITH_UNDERSCORE.

Example 3:

> PROHIBITTED_LISTED

d) The names of data types shall be represented using UpperCamel.

Example 4:

> ResourceHandle

5.2 API Definition

5.2.1 Resource Structure

Resource structure shall define the structure of the resource URIs, the resources, the associated HTTP methods and custom operations used for the service.

Figure 5.2.1-1 provides an example of the resource URI structure (i.e. resource tree) of an API. Table 5.2.1-1 provides an example of an overview of the resources defined for the service, and their applicable HTTP methods and custom operations.

{apiRoot}/<apiName>/<apiVersion>
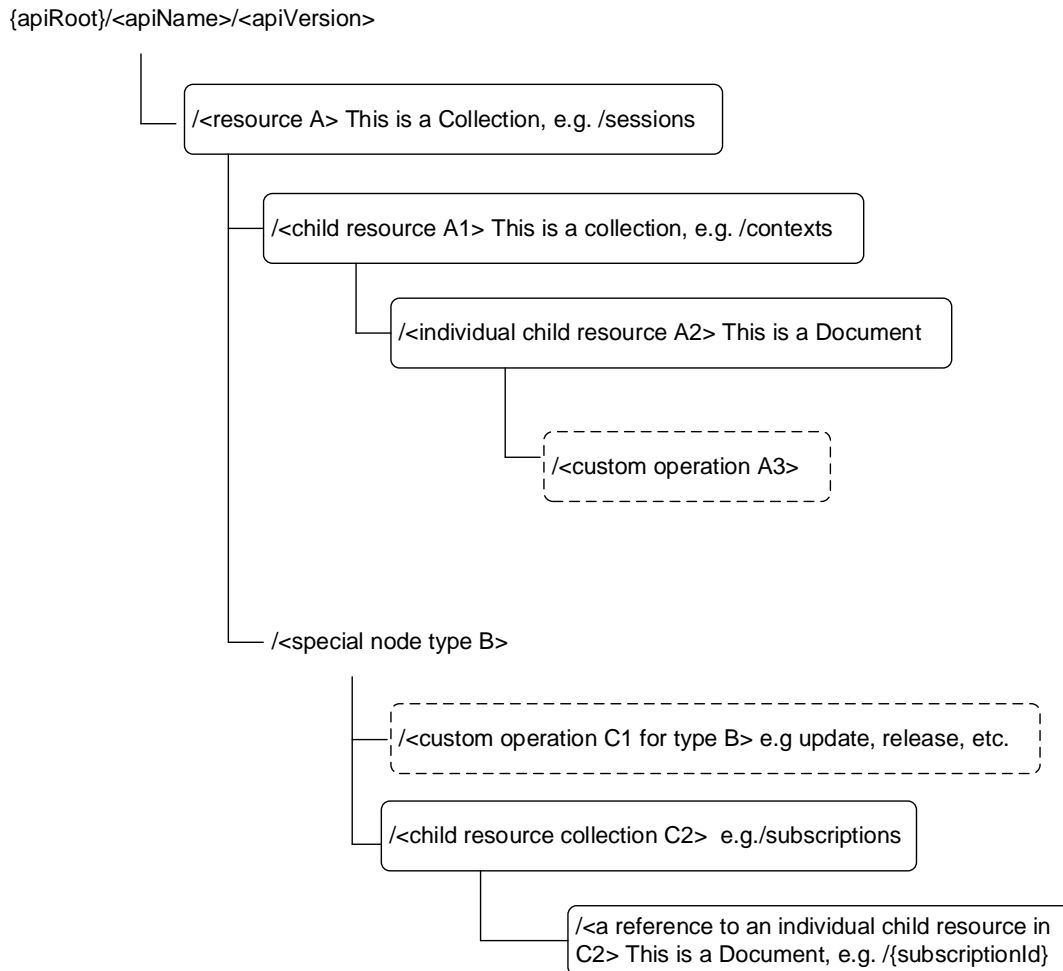
Figure 5.2.1-1: Resource URI structure of the <xyz > API

In figure 5.2.1-1 the following graphical conventions are used:

- a child node with a solid-line frame represents:

    - a resource-URI that has at least one supported HTTP method associated, see e.g. "/resource A" in Figure 5.2.1-1. Such node may be any of the archetypes defined in Annex C1, C2 and C3;

    - a resource-URI, which does not have any standard HTTP operation defined, but has a custom operation, see e.g. "/individual child resource A2" in Figure 5.2.1-1;

- a child node with a dashed-line frame represents a a specific custom operation, see e.g. "/custom operation A3" and "/custom operation C1" in Figure 5.2.1-1. Such node is a Custom operation archetype, see Annex C4;

- a child node without a line around its frame (e.g. "/special node type B" in Figure 5.2.1-1) typically represents a path segment for which no standard HTTP operation is specified. This graphical representation is also used when a custom operation without an associated resource is defined for a path segment. .

Table 5.2.1-1: Resources and methods overview

Resource purpose/name	Resource URI (relative path after API URI)	HTTP method or custom operation	Description (service operation)
<Resource name>	<relative path after API URI>	GET	<Operation executed by GET>
		PUT	<Operation executed by PUT>
		PATCH	<Operation executed by PATCH>
		POST	<Operation executed by POST>
		DELETE	<Operation executed by DELETE>
	<relative path after API URI>/release	release (POST)	<Operation executed by Custom operation>

NOTE 1: The "API URI" is defined in clauses 3.1 and 4.4.1.

For a resource, the "relative path after API URI" is appended to the API URI to form the resource URI as defined in clause 3.1.

5.2.2 Resources and HTTP Methods

Resources and HTTP methods shall specify the resource URI, resource URI variables for the resource and the standard HTTP methods supported by the resource.

Example:

Resource URI: **{apiRoot}/<apiName>/<apiVersion>/<apiSpecificResourceUriPart>**

The resource URI variables supported by the resource shall be defined as table 5.2.2-1 illustrates.

Table 5.2.2-1: Resource URI variables for the resource

Name	Definition
apiRoot	See clause 4.4.1
< Name of resource URI variables for resource>	< Definition of resource URI variables for resource >

The {apiRoot} URI variable should be defined in clauses and this definition should be referenced to ease a possible update of the apiVersion value.

Each method supported by the resource shall be described including the URI query parameters supported by the method, data structures supported by the request body, and the data structures supported by the response body.

URI query parameters supported by the method shall be defined as table 5.2.2-2 illustrates.

Table 5.2.2-2: URI query parameters supported by a method on the resource

Name	Data type	P	Cardinality	Description	Applicability
<name> or n/a	<type> or <leave empty>	<M, C or O>	0..1 or 1 or 0..N or 1..N or <leave empty>	<only if applicable>	

Name: Name of query parameter in URI. If no query parameters are defined for the URI, the name should be marked as "n/a".

Data type: Data type of URI query parameters, i.e. a data type defined in the specification. If no query parameter is defined for the URI, the column is left empty.

P: Presence condition of URI query parameters. It shall be one of "M" (for Mandatory), "C" (for Conditional) and "O" (for Optional). If no query parameter is defined for the URI, the column is left empty.

Cardinality: Defines the allowed number of occurrence. It shall be "0..1", "1", "0..N", "1..N" . If no query parameter is defined for the URI, the column is left empty.

Description: Additional information for URI query parameter, i.e. describes the use of the parameter or the presence condition of the parameter and so on.

Applicability: If the URI query parameter is only applicable for optional feature(s) negotiated using the mechanism defined in clause 6.6.2 of 3GPP TS 29.500 [2], the name of the corresponding feature(s) shall be indicated in this column. If no feature is indicated. the attribute can be used with any feature.

NOTE 1: If no optional features are defined for an API, the applicability column can be omitted for that API.

Data structures supported by the request body of the method shall be specified as table 5.2.2-3 illustrates.

Table 5.2.2-3: Data structures supported by the request body on the resource

Data type	P	Cardinality	Description
"*<type>*" or "array*(<type>)*" or "map*(<type>)*" or n/a	"M", "C" or "O"	"0..1", "1", or "M..N", or <leave empty>	<only if applicable>

Data type: Data type of the data structure in the request body. If the data type is indicated as "*<type>*", the request body shall be of data type *<type>*. If the data type is indicated as "array*(<type>)*", the request body shall be an array (see IETF RFC 8259 [3]) that contains elements of data type *<type>*. If the data type is indicated as "map*(<type>)*", the request body shall be an object (see IETF RFC 8259 [3]) encoding a map (see clause 5.2.4.2) that contains as values elements of data type *<type>*. *<type>* can either be "integer", "number", "string" or "boolean" (as defined in the OpenAPI specification [4]), or a data type defined in a 3GPP specification. If no request body is allowed, the Data type shall be marked as "n/a".

P: Presence condition of a data structure in request body. It shall be one of "M" (for Mandatory), "C" (for Conditional) and "O" (for Optional).

Cardinality: Defines the allowed number of occurrence of data type *<type>*. A cardinality of "*M..N*", is only allowed for data types "array*(<type>)*" and "map*(<type>)*" and indicates the number of elements within the array or map; the values *M* and *N* can either be the characters "M" and "N", respectively, or integer numbers with M being greater than or equal 0, and N being greater than 0 and M. For data type "*<type>*", the cardinality shall be set to "0..1" if the Presence condition is "C" or "O", and to "1" if the Presence condition is "M". The Cardinality shall be left empty if no request body is allowed.

Description: Additional information for a data structure, i.e. describes the use of the data structure or the presence condition of the data structure and so on.

NOTE 2: The cardinality of "0..N" does not imply that the presence condition of the array or map is optional or conditional, i.e. the presence condition can be "M" while the cardinality is "0..N", the presence condition can be "O" or "C" while the cardinality is "1..N".

Data structures supported by the response body of the method shall be specified as table 5.2.2-4 illustrates.

Table 5.2.2-4: Data structures supported by the response body on the resource

Data type	P	Cardinality	Response codes	Description
"*<type>*" or "array*(<type>)*" or "map*(<type>)*" or n/a	"M", "C" or "O"	"0..1", "1", or "M..N", or <leave empty>	<list applicable codes with name from the applicable RFCs>	<Meaning of the success case> or <Meaning of the error case with additional statement regarding error handling>

Data type: Data type of the data structure in the response body. If the data type is indicated as "*<type>*", the response body shall be of data type *<type>*. If the data type is indicated as "array*(<type>)*", the response body shall be an array

(see IETF RFC 8259 [3]) that contains elements of data type *<type>*. If the data type is indicated as "map(*<type>*)", the response body shall be an object (see IETF RFC 8259 [3]) encoding a map (see clause 5.2.4.2) that contains as values elements of data type *<type>*. *<type>* can either be "integer", "number", "string" or "boolean" (as defined in the OpenAPI specification [4]), or a data type defined in a 3GPP specification. If no response body is allowed, the Data type shall be marked as "n/a".

P: Presence condition of a data structure in response body. It shall be one of "M" (for Mandatory), "C" (for Conditional) and "O" (for Optional).

Cardinality: Defines the allowed number of occurrence of data type *<type>*. A cardinality of "*M..N*", is only allowed for data types "array(*<type>*)" and "map(*<type>*)" and indicates the number of elements within the array or map; the values *M* and *N* can either be the characters "M" and "N", respectively, or integer numbers with M being greater than or equal 0, and N being greater than 0 and M. For data type "*<type>*", the cardinality shall be set to "0..1" if the Presence condition is "C" or "O", and to "1" if the Presence condition is "M". The Cardinality shall be left empty if no response body is allowed.

Response codes: Lists applicable response codes with name from HTTP Status Code Registry at IANA [12]. Mandatory HTTP status codes listed in Table 5.2.7.1-1 of 3GPP TS 29.500 [2] for the corresponding HTTP method shall only be included if specific clarifications in the description part or special data types of the response body are required. Applicable HTTP status codes in addition to the mandatory HTTP status codes listed in Table 5.2.7.1-1 of 3GPP TS 29.500 [2] for the corresponding HTTP method shall be included.

Description: Additional information for a response, i.e. describes the meaning of the success case or meaning of the error case with additional statement regarding error handling.

NOTE 3: The cardinality of "0..N" does not imply that the presence condition of the array or map is optional or conditional, i.e. the presence condition can be "M" while the cardinality is "0..N", the presence condition can be "O" or "C" while the cardinality is "1..N".

5.2.3 Representing RPC as Custom Operations on Resources

Custom operations (RPC-style interaction) may be used on a resource. The description of each custom operation contains the custom operation URI, the HTTP method on which the operation is mapped (typically POST), data structures supported by the request body and the data structures supported by the response body.

An overview of the custom operations on a resource is illustrated in table 5.2.3-1.

Table 5.2.3-1: Custom operation

Custom operation URI	Mapped HTTP method	Description
<custom operation URI>	e.g. POST	<Operation executed by custom operation>

Data structures supported by the request body of the method shall be specified as table 5.2.3-2 illustrates.

Table 5.2.3-2: Data structures supported by the mapped HTTP method request body on the resource

Data type	P	Cardinality	Description
"*<type>*" or "array(*<type>*)" or "map(*<type>*)"	"M", "C" or "O"	"0..1", "1", or "M..N", or <leave empty>	<only if applicable>

Data type: Data type of the data structure in the request body. If the data type is indicated as "*<type>*", the request body shall be of data type *<type>*. If the data type is indicated as "array(*<type>*)", the request body shall be an array (see IETF RFC 8259 [3]) that contains elements of data type *<type>*. If the data type is indicated as "map(*<type>*)", the request body shall be an object (see IETF RFC 8259 [3]) encoding a map (see clause 5.2.4.2) that contains as values elements of data type *<type>*. *<type>* can either be "integer", "number", "string" or "boolean" (as defined in the OpenAPI specification [4]), or a data type defined in a 3GPP specification. If no request body is allowed, the Data type shall be marked as "n/a".

P: Presence condition of a data structure in request body. It shall be one of "M" (for Mandatory), "C" (for Conditional) and "O" (for Optional).

Cardinality: Defines the allowed number of occurrence of data type *<type>*. A cardinality of "*M..N*", is only allowed for data types "array(*<type>*)" and "map(*<type>*)" and indicates the number of elements within the array or map; the values *M* and *N* can either be the characters "M" and "N", respectively, or integer numbers with M being greater than or equal 0, and N being greater than 0 and M. For data type "*<type>*", the cardinality shall be set to "0..1" if the Presence condition is "C" or "O", and to "1" if the Presence condition is "M". The Cardinality shall be left empty if no request body is allowed.

Description: Additional information for a data structure, i.e. describes the use of the data structure or the presence condition of the data structure and so on.

NOTE 1: The cardinality of "0..N" does not imply that the presence condition of the array or map is optional or conditional, i.e. the presence condition can be "M" while the cardinality is "0..N", the presence condition can be "O" or "C" while the cardinality is "1..N".

Data structures supported by the response body of the method shall be specified as table 5.2.3-3 illustrates.

Table 5.2.3-3: Data structures supported by the mapped HTTP method response body on the resource

Data type	P	Cardinality	Response codes	Description
"*<type>*" or "array(*<type>*)" or "map(*<type>*)"	"M", "C" or "O"	"0..1", "1" or "M..N", or <leave empty>	<list applicable codes with name from the applicable RFCs>	<Meaning of the success case> or <Meaning of the error case with additional statement regarding error handling>

Data type: Data type of the data structure in the response body. If the data type is indicated as "*<type>*", the response body shall be of data type *<type>*. If the data type is indicated as "array(*<type>*)", the response body shall be an array (see IETF RFC 8259 [3]) that contains elements of data type *<type>*. If the data type is indicated as "map(*<type>*)", the response body shall be an object (see IETF RFC 8259 [3]) encoding a map (see clause 5.2.4.2) that contains as values elements of data type *<type>*. *<type>* can either be "integer", "number", "string" or "boolean" (as defined in the OpenAPI specification [4]), or a data type defined in a 3GPP specification. If no response body is allowed, the Data type shall be marked as "n/a".

P: Presence condition of a data structure in response body. It shall be one of "M" (for Mandatory), "C" (for Conditional) and "O" (for Optional).

Cardinality: Defines the allowed number of occurrence of data type *<type>*. A cardinality of "*M..N*", is only allowed for data types "array(*<type>*)" and "map(*<type>*)" and indicates the number of elements within the array or map; the values *M* and *N* can either be the characters "M" and "N", respectively, or integer numbers with M being greater than or equal 0, and N being greater than 0 and M. For data type "*<type>*", the cardinality shall be set to "0..1" if the Presence condition is "C" or "O", and to "1" if the Presence condition is "M". The Cardinality shall be left empty if no response body is allowed.

Response codes: Lists applicable response codes with name from HTTP Status Code Registry at IANA [12]. Mandatory HTTP status codes listed in Table 5.2.7.1-1 of 3GPP TS 29.500 [2] for the corresponding HTTP method shall only be included if specific clarifications in the description part or special data types of the response body are required. Applicable HTTP status codes in addition to the mandatory HTTP status codes listed in Table 5.2.7.1-1 of 3GPP TS 29.500 [2] for the corresponding HTTP method shall be included.

Description: Additional information for a response, i.e. describes the meaning of the success case or meaning of the error case with additional statement regarding error handling.

NOTE 2: The cardinality of "0..N" does not imply that the presence condition of the array or map is optional or conditional, i.e. the presence condition can be "M" while the cardinality is"0..N", the presence condition can be "O" or "C" while the cardinality is "1..N".

5.2.3A   Notifications

Notifications based on the Callback URI as specified in clause 4.6.2.3 and clause 6.2 of 3GPP TS 29.500 [2] may be supported. The description of each notification contains the Callback URI, the HTTP method (typically POST), data structures supported by the request body and the data structures supported by the response body.

An overview of the notifications is illustrated in table 5.2.3A-1.

Table 5.2.3A-1: Notifications overview

Notification	Callback URI	HTTP method or custom operation	Description (service operation)
<notification 1> e.g. Status Change Notification	< Callback URI > e.g. {StatusCallbackUri}	e.g POST	e.g. Notify Event

Example:

Callback URI: **{notifUri}**

The callback URI variables shall be defined as table 5.2.3A-2 illustrates.

Table 5.2.3A-2: Callback URI variables

Name	Definition
notifUri	String formatted as URI with the Callback Uri

Data structures supported by the request body of the method shall be specified as table 5.2.3A-3 illustrates.

Table 5.2.3A-3: Data structures supported by the HTTP method request body

Data type	P	Cardinality	Description
"<type>" or "array(<type>)" or "map(<type>)"	"M", "C" or "O"	"0..1", "1", or "M..N", or <leave empty>	<only if applicable>

Data type: Data type of the data structure in the request body. If the data type is indicated as "<type>", the request body shall be of data type <type>. If the data type is indicated as "array(<type>)", the request body shall be an array (see IETF RFC 8259 [3]) that contains elements of data type <type>. If the data type is indicated as "map(<type>)", the request body shall be an object (see IETF RFC 8259 [3]) encoding a map (see clause 5.2.4.2) that contains as values elements of data type <type>. <type> can either be "integer", "number", "string" or "boolean" (as defined in the OpenAPI specification [4]), or a data type defined in a 3GPP specification. If no request body is allowed, the Data type shall be marked as "n/a".

P: Presence condition of a data structure in request body. It shall be one of "M" (for Mandatory), "C" (for Conditional) and "O" (for Optional).

Cardinality: Defines the allowed number of occurrence of data type <type>. A cardinality of "*M*..*N*", is only allowed for data types "array(<type>)" and "map(<type>)" and indicates the number of elements within the array or map; the values *M* and *N* can either be the characters "M" and "N", respectively, or integer numbers with M being greater than or equal 0, and N being greater than 0 and M. For data type "<type>", the cardinality shall be set to "0..1" if the Presence condition is "C" or "O", and to "1" if the Presence condition is "M". The Cardinality shall be left empty if no request body is allowed.

Description: Additional information for a data structure, i.e. describes the use of the data structure or the presence condition of the data structure and so on.

NOTE 1: The cardinality of "0..N" does not imply that the presence condition of the array or map is optional or conditional, i.e. the presence condition can be "M" while the cardinality is "0..N", the presence condition can be "O" or "C" while the cardinality is "1..N".

Data structures supported by the response body of the method shall be specified as table 5.2.3A-4 illustrates.

Table 5.2.3A-4: Data structures supported by the mapped HTTP method response body

Data type	P	Cardinality	Response codes	Description
"*<type>*" or "array(*<type>*)" or "map(*<type>*)"	"M", "C" or "O"	"0..1", "1" or "M..N", or <leave empty>	<list applicable codes with name from the applicable RFCs>	<Meaning of the success case> or <Meaning of the error case with additional statement regarding error handling>

Data type: Data type of the data structure in the response body. If the data type is indicated as "*<type>*", the response body shall be of data type *<type>*. If the data type is indicated as "array(*<type>*)", the response body shall be an array (see IETF RFC 8259 [3]) that contains elements of data type *<type>*. If the data type is indicated as "map(*<type>*)", the response body shall be an object (see IETF RFC 8259 [3]) encoding a map (see clause 5.2.4.2) that contains as values elements of data type *<type>*. *<type>* can either be "integer", "number", "string" or "boolean" (as defined in the OpenAPI specification [4]), or a data type defined in a 3GPP specification. If no response body is allowed, the Data type shall be marked as "n/a".

P: Presence condition of a data structure in response body. It shall be one of "M" (for Mandatory), "C" (for Conditional) and "O" (for Optional).

Cardinality: Defines the allowed number of occurrence of data type *<type>*. A cardinality of "*M..N*", is only allowed for data types "array(*<type>*)" and "map(*<type>*)" and indicates the number of elements within the array or map; the values *M* and *N* can either be the characters "M" and "N", respectively, or integer numbers with M being greater than or equal 0, and N being greater than 0 and M. For data type "*<type>*", the cardinality shall be set to "0..1" if the Presence condition is "C" or "O", and to "1" if the Presence condition is "M". The Cardinality shall be left empty if no response body is allowed.

Response codes: Lists applicable response codes with name from HTTP Status Code Registry at IANA [12]. Mandatory HTTP status codes listed in Table 5.2.7.1-1 of 3GPP TS 29.500 [2] for the corresponding HTTP method shall only be included if specific clarifications in the description part or special data types of the response body are required. Applicable HTTP status codes in addition to the mandatory HTTP status codes listed in Table 5.2.7.1-1 of 3GPP TS 29.500 [2] for the corresponding HTTP method shall be included.

Description: Additional information for a response, i.e. describes the meaning of the success case or meaning of the error case with additional statement regarding error handling.

> NOTE 2: The cardinality of "0..N" does not imply that the presence condition of the array or map is optional or conditional, i.e. the presence condition can be "M" while the cardinality is"0..N", the presence condition can be "O" or "C" while the cardinality is "1..N".

5.2.4   Data Models

5.2.4.1   General

The application data model supported by the API shall be specified with the following data types:

1. Structured data types

2. Simple data types

3. Enumerations

4. Binary data

5. Data types describing alternative data types

6. Data types describing combinations of data types

Each data type can be specific for the API or common to multiple APIs (offered by the same or different NFs). The common data types shall be specified in 3GPP TS 29.571 [5].

Characters in an enumeration or string data type in both simple and structured/combination data types specific for the API or common to multiple APIs shall be handled as case sensitive by default, unless stated otherwise.

Strings representing a numerical value or a binary octet string, encoded as hexadecimal characters, shall be handled as case insensitive.

5.2.4.2   Structured data types

The structured data types shall represent an object (see IETF RFC 8259 [3]). The structured data types shall contain attributes that are simple data types, structured data types, arrays (see below), maps (as defined below), enumerations, data types describing alternative data types, data types describing combinations of data types or "Any Type" (as described below).

An array (see IETF RFC 8259 [3]) shall represent a list of values without keys and with significance in the order of sequence. All values shall be of the same type.

A map shall represent an object (see IETF RFC 8259 [3]) with a list of key-value pairs (with no significance in the order of sequence), where all keys are of type string and shall be unique identifiers assigned by the application rather than by the schema, and where all values shall be of the same type.

NOTE 1: Maps are supported by the OpenAPI specification [4] as described at https://swagger.io/docs/specification/data-models/dictionaries/. Maps can enable a faster lookup of elements identified by some key in huge data structures compared to arrays that contain the key within the elements. Maps can also be used instead of arrays to modify individual elements when modification instructions of the PATCH method are compliant to IETF RFC 7396 [7].

Each structured data type shall be specified in a separate clause as illustrated in table 5.2.4.2-1.

Table 5.2.4.2-1: Definition of type <Data type>

Attribute name	Data type	P	Cardinality	Description	Applicability
<attribute name>	"*<type>*" or "array*(<type>)*" or "map*(<type>)*" or "Any Type"	"M", "C" or "O"	"0..1", "1" or "M..N"	<only if applicable>	

Table 5.2.4.2-2: Definition of type <Data type> with nested structures

Attribute name	Data type	P	Cardinality	Description	Applicability
<attribute name>	"array*(map(<type>)*)" or "map*(array(<type>)*)" or" array*(array(<type>)*)" or "map*(map(<type>)*)"	"M", "C" or "O"	"M1..N1(M2..N2)"	<only if applicable>	

Attribute name: Name of attributes that belong to the specified data type. The attribute names within a structured data type shall be unique, and their relative order inside the structured data type shall not imply any specific ordering of the corresponding JSON elements in a JSON object.

NOTE 2: The JSON specification (IETF RFC 8259 [3]) allows duplicate keys in a JSON object, but its usage is discouraged, since it makes interoperability more difficult, and makes the behavior of the JSON parsing software unpredictable. Similarly, that RFC encourages to make implementations not dependent on attribute ordering.

Data type: Data type of the attribute. If the data type is indicated as "*<type>*", the attribute shall be of data type *<type>*. If the data type is indicated as "array*(<type>)*", the attribute shall be an array (see IETF RFC 8259 [3]) that contains elements of data type *<type>*. If the data type is indicated as "map*(<type>)*", the attribute shall be an object (see IETF RFC 8259 [3]) encoding a map that contains as values elements of data type *<type>*. *<type>* can either be "integer", "number", "string" or "boolean" (as defined in the OpenAPI specification [4]), or a data type defined in a

3GPP specification. If the data type is indicated as "Any Type", the attribute can either be "integer", "number", "string", "boolean", "array" or "object" (as defined in the OpenAPI specification [4]), or a data type defined in a 3GPP specification.

P: Presence condition of a data structure in request body. It shall be one of "M" (for Mandatory), "C" (for Conditional) and "O" (for Optional).

Cardinality: Defines the allowed number of occurrence of data type <type>. A cardinality of "M..N", is only allowed for data types "array(<type>)" and "map(<type>)" and indicates the number of elements within the array or map; the values M and N can either be the characters "M" and "N", respectively, or integer numbers; with M being greater than or equal 0, and N being greater than 0 and M. For data type with nested structures, the cardinality is expressed in a nested manner, each level of the M and N values describe the cardinality of the corresponding level of the data structure. For data type "<type>" and "Any Type", the cardinality shall be set to "0..1" if the Presence condition is "C" or "O", and to "1" if the Presence condition is "M".

Description: Describes the meaning and use of the attribute and may contain normative statements.

Applicability: If the attribute is only applicable for optional feature(s) negotiated using the mechanism defined in clause 6.6 of 3GPP TS 29.500 [2], the name of the corresponding feature(s) shall be indicated in this column. If no feature is indicated. the attribute can be used with any feature.

NOTE 3: The cardinality of "0..N" does not imply that the presence condition of the array or map is optional or conditional, i.e. the presence condition can be "M" while the cardinality is "0..N", the presence condition can be "O" or "C" while the cardinality is "1..N".

NOTE 4: If no optional features are defined for an API, the applicability column can be omitted for that API.

5.2.4.3 Simple data types and enumerations

The simple data types and enumerations can be referenced from data structures. All simple data types and enumerations should map to base types supported by OpenAPI specification [4]. Simple data types shall be specified as illustrated in table 5.2.4.3-1.

Table 5.2.4.3-1: Simple data types

Type Name	Type Definition	Description	Applicability
	<one simple data type, i.e. boolean, integer, number, or string>		

Type Name: The name of the simple data type.

Type Definition: Base types supported by OpenAPI specification [4], i.e. Boolean, integer, string and so on.

Description: Additional descriptions for simple data types like range, string length, string pattern (regular expression) and so on.

Applicability: If the type is only applicable for optional feature(s) negotiated using the mechanism defined in clause 6.6 of 3GPP TS 29.500 [2], the name of the corresponding feature(s) shall be indicated in this column. If no feature is indicated. the type can be used with any feature.

NOTE 1: If no optional features are defined for an API, the applicability column can be omitted for that API.

Each enumeration type shall be respectively specified for their elements sets as illustrated in table 5.2.4.3-2.

Table 5.2.4.3-2: Enumeration < EnumType>

Enumeration value	Description	Applicability
Enumeration value 1	The description of this enum value	
Enumeration value 2	The description of this other enum value	

Enumeration value: Defines the valid values, which can be integer, string or boolean, it is suggested to keep the same value style in one API specification.

Description: Additional descriptions for enumeration attributes, like the meaning and usage of enumeration elements.

Applicability: If the enumeration value is only applicable for optional feature(s) negotiated using the mechanism defined in clause 6.6 of 3GPP TS 29.500 [2], the name of the corresponding feature(s) shall be indicated in this column. If no feature is indicated. the enumeration value can be used with any feature.

NOTE 2: If no optional features are defined for an API, the applicability column can be omitted for that API.

5.2.4.4 Binary Data

5.2.4.5 Data types describing alternative data types or combinations of data types

The data types describing alternative data types or combinations of data types shall represent an OpenAPI schema object using the "oneOf" , "anyOf" or "allOf" keyword to list alternative or to be combined data types (see the OpenAPI specification [4] and https://swagger.io/docs/specification/data-models/oneof-anyof-allof-not/).

An instance (i.e. a corresponding part of a JSON file to be evaluated against the schema) matches a list of mutually exclusive alternative data types , as described using the OpenAPI "oneOf" keyword, if the instance matches against one and only one of the alternative data types.

NOTE 1: Data types with the same simple data type but different format and/or pattern attributes are mutually exclusive if the corresponding formats and/or patterns are mutually exclusive.

An instance (i.e. a corresponding part of a JSON file to be evaluated against the schema) matches a list of non-exclusive alternative data types, as described using the OpenAPI "anyOf" keyword, if the instance matches against one or more of the alternative data types.

An instance (i.e. a corresponding part of a JSON file to be evaluated against the schema) matches a list of to be combined data types, as described using the OpenAPI "allOf" keyword, if the instance matches against all of the to be combined data types.

The alternative or to be combined data types shall be simple data types, structured data types, arrays (see clause 5.2.4.2), maps (see clause 5.2.4.2), enumerations, data types describing alternative data types, or data types describing combinations of data types.

Each structured data type shall be specified in a separate clause as illustrated in table 5.2.4.2-1.

Table 5.2.4.2-1: Definition of type <Data type> as a list of <"mutually exclusive alternatives" / "non-exclusive alternatives" / "to be combined data types">

Data type	Cardinality	Description	Applicability
"*<type>*" or "array*(<type>)*" or "map*(<type>)*"	"1" or "*M..N*"	<only if applicable>	

Data type: Data type of the alternative or to be combined data type. If the data type is indicated as "*<type>*", the alternative or to be combined data type shall be of data type *<type>*. If the data type is indicated as "array(*<type>*)", the alternative or to be combined data type shall be an array (see IETF RFC 8259 [3]) that contains elements of data type *<type>*. If the data type is indicated as "map(*<type>*)", the alternative or to be combined data type shall be an object (see IETF RFC 8259 [3]) encoding a map (see clause 5.2.4.2) that contains as values elements of data type *<type>*. *<type>* can either be "integer", "number", "string" or "boolean" (as defined in the OpenAPI specification [4]), or a data type defined in a 3GPP specification.

Cardinality: Defines the allowed number of occurrence of data type <type>. A cardinality of "M..N", is only allowed for data types "array(<type>)" and "map(<type>)" and indicates the number of elements within the array or map; the values M and N can either be the characters "M" and "N", respectively, or integer numbers; with M being greater than or equal 0, and N being greater than 0. For data type "<type>", the cardinality shall be set to "1".

Description: Describes the meaning and use of the attribute and may contain normative statements.

Applicability: If the type is only applicable for optional feature(s) negotiated using the mechanism defined in clause 6.6 of 3GPP TS 29.500 [2], the name of the corresponding feature(s) shall be indicated in this column. If no feature is indicated. the type can be used with any feature.

NOTE 2: If no optional features are defined for an API, the applicability column can be omitted for that API.

NOTE 3: Data types describing alternative data types or combinations of data types can only be extended with additional data types in a backward compatible manner if the new data types are associated with an optional feature and the mechanism defined in clause 6.6 of 3GPP TS 29.500 [2] is used to negotiate the support of that optional feature before that new data type is used.

5.2.5 Void

5.2.6 3GPP Custom HTTP Headers

The definition of 3GPP APIs for the Service-Based Architecture includes the definition of 3GPP-specific HTTP headers, both generic (i.e. applicable to any API), and also API-specific headers.

The definition of the generic HTTP headers is done in 3GPP TS 29.500 [2]. It consists of the ABNF definition of each header, along with a detailed description of the syntax and expected behaviour and handling by the receiver of such header; each header is specified in its own separate clause.

3GPP TS 29.500 [2], Annex D, contains a self-contained full grammar of the entire set of custom HTTP headers, comprising both the definition of ABNF components/rules re-used from IETF RFCs, and the 3GPP-specific ABNF rule definitions.

In addition, the zip file of the published 3GPP TS 29.500 [2] shall contain a file, named "TS29500_CustomHeaders.abnf", that reflects the content of such Annex D, and the same file shall be made available on the 3GPP Forge GitLab repository mentioned in clause 5.3.1, along with the OpenAPI files, where tooling can be used for validation of the ABNF grammar.

For API-specific HTTP header definition, a similar procedure shall be followed: if an API defines custom HTTP headers, their ABNF shall be included in the TS where such API is defined, along with the description of each header; each header shall be described in its own separate clause.

Similarly, each TS that defines custom HTTP headers shall contain an Annex with a self-contained and minimal set of ABNF rules, including re-used rules from other sources (e.g. IETF RFCs, 3GPP TS 29.500, etc.). The zip file of the published TS shall contain a file with ".abnf" extension, with the same content as the Annex, and such file shall be made available, and validated, on 3GPP Forge/GitLab.

5.3 OpenAPI specification files

5.3.1 General

5GC SBI APIs' OpenAPI specification files shall comply with the OpenAPI specification [4] and with the present clause 5.3.

Each API shall be described in one OpenAPI specification file contained in an Annex of the 3GPP specification that describes the corresponding API. In addition, 3GPP specifications may contain OpenAPI specification file with common data types.

Informative copies of all OpenAPI specification files contained in 3GPP Technical Specifications are available on a Git-based repository hosted in 3GPP Forge, that uses the GitLab software version control system.

The repository is named "5G APIs" and is publicly accessible via the following URI:

https://forge.3gpp.org/rep/all/5G_APIs

NOTE: As indicated in 3GPP TR 21.900 [25], clause 5C, there is an alternative option for the availability and distribution of OpenAPI specification files (decided by the responsible Working Group of each TS), where these files can be stored in a 3GPP Forge GitLab repository and can be declared as normative.

In that case, the TS does not contain OpenAPI descriptions in normative Annexes and instead it includes information about the repository used for the storage of OpenAPI files, e.g., GitLab group, repository name, branch, directory path (if any), and a Git tag representing the repository snapshot that corresponds to the specific TS version.

5.3.2   Formatting of OpenAPI specification files

The following guidelines shall be used when documenting OpenAPI specification files:

-   OpenAPI specification files shall be documented using YAML format (see YAML 1.2 [16]). For specific restrictions on the usage of YAML in OpenAPI, see OpenAPI Specification [4].

-   The style used for the specification shall be "PL" (Programming Language).

-   The different scopes in the YAML data structures representing collections (objects, arrays…) shall use an indentation of two white spaces.

-   Comments may be added by following the standard YAML syntax ("#").

-   Tabs shall not be used in OpenAPI specification files (e.g. within description fields).

-   "Unbreakable" spaces (UTF-8 'NO-BREAK SPACE' (U+00A0)) shall not be used in OpenAPI specification files (e.g. within description fields). Only "normal" spaces (UTF-8 'SPACE' (U+0020)) shall be allowed.

-   Trailing spaces (i.e. white spaces at the end of a line) should not be used in OpenAPI specification files.

5.3.3   Info

The OpenAPI specification file of an API shall contain an "info" object with the title that should be set to the same value as chosen for the API name in the heading of Annex A.x of the corresponding 3GPP TS, and with the version set as described in clause 4.3.

The "info" object shall also include a "description" field, containing the name of the service that the API implements, and the same copyright notice as included in the front page of the corresponding 3GPP TS that specifies the API. The content of the "description" field shall be formatted using the YAML block notation for scalars (i.e. using the "|" character).

EXAMPLE: "info" object with the title, version and description of the API.

```
info:
  title: 'Nsmf_PDUSession'
  version: '1.0.0'
  description: |
    SMF PDUSession Service.
    © 2020, 3GPP Organizational Partners (ARIB, ATIS, CCSA, ETSI, TSDSI, TTA, TTC).
    All rights reserved.
```

5.3.4   externalDocs

Each OpenAPI specification file shall provide an "externalDocs" object as illustrated in the example below that shall contain:

-   within the "description" field the 3GPP TS number, the version number and the name of the 3GPP TS describing the API, and

-   within the "url" field a reference to the folder of that TS within the specification archive of the public 3GPP fileserver (i.e. "https://www.3gpp.org/ftp/Specs/archive/<specSeries>/<SpecNumber>/").

The version number in the "externalDocs" object shall be updated each time when the TS version contains new changes to the OpenAPI specification file.

NOTE 1: If a new TS version is provided without any changes to the OpenAPI specification file, the TS version number included in the "description" field of the "externalDocs" field in the OpenAPI specification file is not updated.

NOTE 2: If a new TS version is provided with changes to the OpenAPI specification file, the TS version number included in the "description" field of the "externalDocs" object is updated. However, if the API version in a given release is still the same as in a previous release, the first TS version in the first release that contains that API version is provided as TS version within the "externalDocs" object also in the TS versions of the subsequent releases.

EXAMPLE: "externalDocs" object.

```
externalDocs:
  description: 3GPP TS 29.503 V15.1.0; 5G System; Unified Data Management Services
  url: 'http://www.3gpp.org/ftp/Specs/archive/29_series/29.503/'
```

5.3.5   Servers

As defined in clause 4.4.1, the API URI consists of **{apiRoot}/<apiName>/<apiVersion>**. It shall be encoded in the corresponding OpenAPI specification file as "servers" field with **{apiRoot}** as variable.

Example:

```
servers:
  - url: '{apiRoot}/nxxx-yyyy/v1'
    variables:
      apiRoot:
        default: https://example.com
        description: apiRoot as defined in clause 4.4.1 of 3GPP TS 29.501
```

5.3.6   References to other 3GPP-defined OpenAPI specification files

Open API specification files may contain references to fragments of other 3GPP-defined Open API specification files.

Such references shall be formatted to refer to local files stored on the same folder.

NOTE 1: For the purpose of referencing, it is assumed that each OpenAPI specification file contained in a 3GPP specification is stored as separate physical file and that all OpenAPI specification files are stored in the same directory on the local server.

The referenced file names for other 3GPP-defined Open API specification files shall comply with the following convention, unless a specific file name is indicated in the Annex of a 3GPP specification defining an OpenAPI specification file. The file name shall consist of (in the order below):

- the 3GPP specification number in the format "TSxxyyy";

- an "_" character;

- if the OpenAPI specification file contains an API definition: the API name which shall be taken from the heading of the relevant annex A.x as defined in the corresponding 3GPP TS of that API.

- if the OpenAPI specification file contains a definition of CommonData: the string "CommonData"; and

- the string ".yaml".

NOTE 2: The informative copies of OpenAPI specification files contained in 3GPP Technical Specifications at the public 3GPP file server (see clause 5.3.1) follow the above conventions and can be copied into a local folder in order to resolve references.

Such a reference to another OpenAPI specification file shall be interpreted as refering to the related OpenAPI specification file contained in the version of the corresponding 3GPP TS indicated in the reference clause of the specification, i.e. for a non-specific reference the latest version of that 3GPP TS in the same Release as the specification.

EXAMPLE: Reference to Data Type "Xxx" defined in the same file

```
$ref: '#/components/schemas/Xxx'
```

EXAMPLE: Reference to Data Type "Xxx" defined as Common Data in 3GPP TS 29.571:

```
$ref: 'TS29571_CommonData.yaml#/components/schemas/Xxx'
```

EXAMPLE: Reference to Data Type "Xxx" defined within API "Nudm_UEAU" in 3GPP "TS 29.503":

```
$ref: 'TS29503_Nudm_UEAU.yaml#/components/schemas/Xxx'
```

5.3.7 Server-initiated communication

If an API contains server-initiated communication (see clause 6.2 of 3GPP TS 29.500 [2]), e.g. for notifications as described in clause 4.6.2.3, it should be described as "callbacks" in OpenAPI specification files.

Example:

```
paths:
  /subscriptions:
    post:
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                callbackUrl: # Callback URL
                  type: string
                  format: uri
      responses:
        '201':
          description: Success
      callbacks:
        myNotification: # arbitrary name
          '{$request.body#/callbackUrl}': # refers The callback URL in the POST
            post:
              requestBody: # Contents of the callback message
                required: true
                content:
                  application/json:
                    schema:
                      $ref: '#/components/schemas/NotificationBody'
              responses: # Expected responses to the callback message
                '204':
                  description: xxx
```

An API description may consist of standalone notifications, i.e., notifications where there is no previous interaction between consumer and producer to create a subscription. In such case, the API shall be documented by including a "dummy" subscription request pseudo-operation that shall not be used by the consumer. In such case, the notification URI may be obtained by the producer by different means (e.g. out of band configuration, or via a separate signaling interaction with the producer, or by querying NRF for the NF Profile of the consumer and looking for a "default notification endpoint").

Example:

```
paths:
  /subscriptions:
    post:
      # This is a pseudo operation, consumers shall NOT invoke this method!
```

```
    requestBody:
      required: true
      content:
        application/json:
          # Unspecified schema for the JSON body, since this is not used by consumer or producer
          schema: {}
    responses:
      default:
        $ref: 'TS29571_CommonData.yaml#/components/responses/default'
    callbacks:
      standaloneNotification:
        '{notificationUri}':
        # The URI in {notificationUri} is obtained out of band by the producer
          post:
            requestBody: # Contents of the callback message
              required: true
              content:
                application/json:
                  schema:
                    $ref: '#/components/schemas/NotificationBody'
            responses: # Expected responses to the callback message
              '204':
                description: xxx
```

5.3.8   Describing the body of HTTP PATCH requests

5.3.8.1   General

As described in clause 4.6.1.1.3.2, options to use for bodies of HTTP PATCH requests are

- A "JSON Merge Patch" encoding as defined in IETF RFC 7396 [7].

- A "JSON Patch" encoding as defined IETF RFC 6902 [8].

- A HTTP multipart messages with the multipart/mixed content-type as defined in IETF RFC 2046 [29].

It is possible to allow the three encodings in a OpenAPI Specification [4] offering the three schemas as alternative contents.

  NOTE: In Rel-15 a single encoding will be selected for each resource as backward compatibility considerations do not yet apply.

An example OpenAPI specification file offering both PATCH encodings is included in Annex D.

5.3.8.2   JSON Merge Patch

In the OpenAPI Specification [4] file, the content field key of the Request Body Object shall contain "application/merge-patch+json". The content field value is a Media Type Object identifying the applicable patch body Schema Object. The patch body Schema Object may contain structured data types derived from the data types used in the schema to describe a complete representation of the resource in such a manner that attributes that are allowed to be modified are listed in the "properties" validation keyword.

  NOTE 1: A derived structured data type is beneficial if the data types used to describe a complete representation of the resource contains mandatory attributes, if attributes are allowed to be removed by the PATCH operation, or if a checking by the OpenAPI tooling that only allowed modifications are done via the "additionalProperties: false" keyword is desired. It also provides a clear description in the OpenAPI specification file to developers which modifications need to be supported.

As an alternative, the data types used in the schema to describe a complete representation of the resource may be used if any attributes that are allowed to be removed are marked as "nullable: true" in that schema.

Any attributes that are allowed to be removed shall be marked as "nullable: true" in the patch body Schema Object.

The "additionalProperties: false" keyword may be set.

NOTE 2: The "additionalProperties: false" keyword enables the OpenAPI tooling to check that only allowed modifications are done. Extensions of the object in future releases are still possible under the assumption that the supported features mechanism is used to negotiate the usage of any new attribute prior to the PATCH invocation. If new optional attributes are expected to be introduced without corresponding supported feature or if PATCH can be used as first operation in an API, the usage of the "additionalProperties: false" keyword is not appropriate.

5.3.8.3 JSON PATCH

In the OpenAPI Specification [4] file, the content field of the key Request Body Object shall contain "application/json-patch+json". The content field value is a Media Type Object identifying the applicable patch body. It may contain a mutually exclusive list (using the "oneOf" keyword) of all allowed modifications as <path, op, value> tuples, where "path" is a string containing a JSON Pointer value referring to a JSON object that is allowed to be modified, "op" is an enumeration of allowed JSON PATCH operations on the JSON object identified by "path" and "value" representing the schema/type of the value that will be updated or added at the JSON object identified by "path". In addition, an open alternative containing an object with no properties may be added using the "anyOf" keyword.

NOTE 1: A mutually exclusive list provides a clear description in the OpenAPI specification file to developers which modifications need to be supported. This is of particular interest if only a limited number of modifications need to be supported. If no open alternative is included, the OpenAPI tooling will also check that only allowed modifications are done.

NOTE 2: The open alternative allows for extensions of the PATCH in scenarios where new optional attributes are expected to be introduced without corresponding supported feature or if PATCH can be used as first operation in an API.

5.3.8.4 Multipart Messages

In the OpenAPI Specification [4] file, the content field key of the Request Body Object shall contain "multipart/mixed". The content field value is a Media Type Object identifying the applicable patch body Schema Object.

The resource is encoded as an HTTP multipart message with the multipart/mixed content-type as described in IETF RFC 2046 [29].

The boundary parameter is used to delimit each part (Start of first body part, second body part, etc) and shall be set to a value as in accordance with IETF RFC 2046 [29].

The JSON body part will always be inserted as the first body part of the message and shall adhere to the JSON document structure defined in IETF RFC 6902 [14], the Content-Id of the first body part shall be "patch" and the content type shall be "application/json-patch+json". The content field value of the first body part is a Media Type Object identifying the applicable patch body and encoded as specified in clause 5.3.8.3.

Use of multipart messages in HTTP PATCH is documented in specific specifications.

5.3.9 Structured data types

For a structured data type, as defined in clause 5.2.4.2, the OpenAPI Specification [4] file shall contain a definition in the components/schemas clause defining a schema with the name of the structured data type as key.

The schema shall contain:

- "type: object";

- "description: *<description>*", where *<description>* is the description of the data type in the table defining the structured data type. The "description" attribute should be provided for all data types, specially if they are frequently reused from the same or other OpenAPI specification files; the "description" attribute shall always be provided for data types defined as maps, with a clear indication of the values (strings) used as key of the map.

- if any attributes in the structured data type are marked as mandatory, a "required" keyword listing those attributes; and

- a "properties" keyword containing for each attribute in the structured data type an entry with the attribute name as key and:

1. if the data type is "*<type>*":

    a. if the data type of the attribute is "string", "number", "integer", or "boolean";

        i) a type definition using that data type as value ("type: *<data type>*"); and

        ii) optionally "description: *<description>*", where *<description>* is the description of the attribute in the table defining the structured data type; or

    b. otherwise a reference to the data type schema for the data type *<data type>* of the attribute, i.e. "$ref: '#/components/schemas/*<data type>*'" if that data type schema is contained in the same OpenAPI specification file and "$ref: '*<filename>*#/components/schemas/*<data type>*'" if that data type schema is contained in file *<filename>* in the same directory on the same server;

2. if the data type is "array*(<type>)*":

    a. a type definition "type: array";

    b. an "items:" definition containing:

        i). if the data type of the attribute is "string", "number", "integer", or "boolean", a type definition using that data type as value ("type: *<data type>*"); or

        ii). otherwise a reference to the data type schema for the data type *<data type>* of the attribute, i.e. "$ref: '#/components/schemas/*<data type>*'" if that data type schema is contained in the same OpenAPI specification file and "$ref: '*<filename>*#/components/schemas/*<data type>*'" if that data type schema is contained in file *<filename>* in the same directory on the same server;

    c. if the cardinality contained an integer value *<m>* as lower boundary, "minItems: *<m>*"; and

    d. if the cardinality contained an integer value *<n>* as upper boundary, "maxItems: *<n>*"; and

    e. optionally "description: *<description>*", where *<description>* is the description of the attribute in the table defining the structured data type;

3. if the data type is "map*(<type>)*":

    a. a type definition "type: object"; and

    b. an "additionalProperties:" definition containing:

        i). if the data type of the attribute is "string", "number", "integer", or "boolean", a type definition using that data type as value ("type: *<data type>*"); or

        ii). otherwise a reference to the data type schema for the data type *<data type>* of the attribute, i.e. "$ref: '#/components/schemas/*<data type>*'" if that data type schema is contained in the same OpenAPI specification file and "$ref: '*<filename>*#/components/schemas/*<data type>*'" if that data type schema is contained in file *<filename>* in the same directory on the same server;

    c. if the cardinality contained an integer value *<m>* as lower boundary, "minProperties: *<m>*"; and

    d. if the cardinality contained an integer value *<n>* as upper boundary, "maxProperties: *<n>*"; and

    e. "description: *<description>*", where *<description>* is the description of the attribute in the table defining the structured data type; the "description" attribute shall always be provided for attributes defined as maps, with a clear indication of the values (strings) used as key of the map.

NOTE: An omission of the "minProperties", and "maxProperties" keywords indicates that no lower or upper boundaries respectively, for the number of properties in an object are defined. An omission of the "minItems", and "maxItems" keywords indicates that no lower or upper boundaries, respectively, for the number of items in an array are defined.

4. if the data type is "Any Type":

    a. if no properties to be defined, a pair of curly braces after the attribute name key "*<attribute name>*: {}", which is shorthand syntax for an arbitrary-type schema; or

b. at least one of the following properties:

    i) if null value is allowed, "nullable: true"; or

    ii). "description: <description>", where <description> is the description of the attribute in the table defining the structured data type.

In the OpenAPI 3.0.0 Specification [4], the '$ref' keyword must be the only attribute of the JSON object. Therefore it is not allowed to use e.g. a "description" attribute along with the '$ref' keyword. However, if it is deemed necessary, a "description" attribute can be added before or after the '$ref' keyword as a comment, i.e. by adding a comment sign '#' in front of the "description" field.

NOTE: CT4 decides on case-by-case basis when it is deemed necessary to add such a "description" attribute along with the '$ref' keyword.

Example:

Table 5.3.9-1: Definition of type ExampleStructuredType

Attribute name	Data type	P	Cardinality	Description	Applicability
exSimple	ExSimple	M	1	exSimple attribute description	
exArrayElements	array(string)	O	0..10	exArrayElements attribute description	
exMapElements	map(ExStructure)	M	1..N	exMapElements attribute description	
exNestedArray	array(map(string))	M	0..N(1..M)	exNestedArray attribute description	
exNestedMap	map(array(string))	O	1..N(2..M)	exNestedMap attribute description	
exAnyTypeNullableElement	Any Type	O	0..1	exAnyTypeNullableElement attribute description	
exAnyTypeNoDescription	Any Type	O	0..1	n/a	

The data structure in table 5.3.9-1 is described in an OpenAPI specification file as follows:

```
components:
  schemas:
    ExampleStructuredType:
      type: object
      description: ExampleStructuredType data type description
      required:
        - exSimple
        - exMapElements
      properties:
        exSimple:
          $ref: '#/components/schemas/ExSimple'
        exArrayElements:
          type: array
          items:
            type: string
          minItems: 0
          maxItems: 10
          description: exArrayElements attribute description
        exMapElements:
          type: object
          additionalProperties:
            # description: commented out description of the additionalProperties
            $ref: '#/components/schemas/ExStructure'
          minProperties: 1
          description: exMapElements attribute description, indicating values of the map keys
        exNestedArray:
          type: array
          items:
            type: object
            additionalProperties:
              type: string
            minProperties: 1
          description: exNestedArray attribute description
        exNestedMap:
          type: object
          additionalProperties:
            type: array
            items:
              type: string
```

```
        minItems: 2
      minProperties: 1
      description: exNestedMap attribute description
    exAnyTypeNullableElement:
      nullable: true
      description: exAnyTypeNullableElement attribute description
    exAnyTypeNoDescription: {}
```

5.3.10 Data types describing alternative data types or combinations of data types

For a data type describing alternatives, as defined in clause 5.2.4.5, the OpenAPI Specification [4] file shall contain a definition in the components/schemas clause defining a schema with the name of the data type describing alternatives as key.

The schema shall contain:

- the "oneOf", "anyOf" or "allOf" keyword selected as follows:

   1. for table caption "Definition of type <Data type> as a list of mutually exclusive alternatives", the "oneOf" keyword;

   2. for table caption "Definition of type <Data type> as a list of non-exclusive alternatives", the "anyOf" keyword;

   3. for table caption "Definition of type <Data type> as a list of to be combined data types", the "allOf" keyword;

- a list of alternatives, containing for each line in the table describing the data type a separate line starting with "-":

   1. if the data type is "*<type>*":

      a. if the data type of the attribute is "string", "number", "integer", or "boolean";

         i) a type definition using that data type as value ("type: *<data type>*"); and

         ii) optionally "description: *<description>*", where *<description>* is the description of the attribute in the table defining the structured data type; or

      b. otherwise a reference to the data type schema for the data type *<data type>* of the attribute, i.e. "$ref: '#/components/schemas/*<data type>*'" if that data type schema is contained in the same OpenAPI specification file and "$ref: '*<filename>*#/components/schemas/*<data type>*'" if that data type schema is contained in file *<filename>* in the same directory on the same server;

   2. if the data type is "array*(<type>)*":

      a. a type definition "type: array";

      b. an "items:" definition containing:

         i). if the data type of the attribute is "string", "number", "integer", or "boolean", a type definition using that data type as value ("type: *<data type>*"); or

         ii). otherwise a reference to the data type schema for the data type *<data type>* of the attribute, i.e. "$ref: '#/components/schemas/*<data type>*'" if that data type schema is contained in the same OpenAPI specification file and "$ref: '*<filename>*#/components/schemas/*<data type>*'" if that data type schema is contained in file *<filename>* in the same directory on the same server;

      c. if the cardinality contained an integer value *<m>* as lower boundary, "minItems: *<m>*"; and

      d. if the cardinality contained an integer value *<n>* as upper boundary, "maxItems: *<n>*"; and

      e. optionally "description: *<description>*", where *<description>* is the description of the attribute in the table defining the structured data type;

   3. if the data type is "map*(<type>)*":

    a. a type definition "type: object"; and

    b. an "additionalProperties:" definition containing:

        i). if the data type of the attribute is "string", "number", "integer", or "boolean", a type definition using that data type as value ("type: *\<data type\>*"); or

        ii). otherwise a reference to the data type schema for the data type *\<data type\>* of the attribute, i.e. "$ref: '#/components/schemas/*\<data type\>*'" if that data type schema is contained in the same OpenAPI specification file and "$ref: '*\<filename\>*#/components/schemas/*\<data type\>*'" if that data type schema is contained in file *\<filename\>* in the same directory on the same server;

    c. if the cardinality contained an integer value *\<m\>* as lower boundary, "minProperties: *\<m\>*"; and

    d. if the cardinality contained an integer value *\<n\>* as upper boundary, "maxProperties: *\<n\>*"; and

    e. optionally "description: *\<description\>*", where *\<description\>* is the description of the attribute in the table defining the structured data type.

NOTE: An omission of the "minProperties", and "maxProperties" keywords indicates that no lower or upper boundaries respectively, for the number of properties in an object are defined. An omission of the "minItems", and "maxItems" keywords indicates that no lower or upper boundaries, respectively, for the number of items in an array are defined.

Example:

Table 5.3.10-1: Definition of type ExampleAlternativesType as a list of mutually exclusive alternatives

Data type	Cardinality	Description	Applicability
ExSimple	1	exSimple attribute description	
array(string)	0..10	exArrayElements attribute description	
map(ExStructure)	1..N	exMapElements attribute description	

The data structure in table 5.3.10-1 is described in an OpenAPI specification file as follows:

```
components:
  schemas:
    ExampleAlternativesType:
      oneOf:
      - $ref: '#/components/schemas/ExSimple'
      - type: array
        items:
          type: string
        minItems: 0
        maxItems: 10
        description: exArrayElements attribute description
      - type: object
        additionalProperties:
          $ref: '#/components/schemas/ExStructure'
        minProperties: 1
        description: exMapElements attribute description
```

5.3.11 Error Responses

As described in clause 4.8 of the present specification and clause 5.2.7 of 3GPP TS 29.500 [2], 5GC SBI APIs use valid HTTP response codes as error codes in HTTP responses and may include a "ProblemDetails" data structure specified in clause 5.2.4.1 of 3GPP TS 29.571 [5] or an application-specific data structure.

Table 5.2.7.1-1 of 3GPP TS 29.500 [2] specifies HTTP status code per HTTP method. For each HTTP method of an API, HTTP status codes shall be specified using response code tables as described in clauses 5.2.2 and 5.2.3. OpenAPI specification files shall include in the description of an HTTP method in the "paths" segment the mandatory HTTP status codes in Table 5.2.7.1-1 of 3GPP TS 29.500 [2] and the HTTP status codes listed in response codes table of that HTTP method.

For the purpose of referencing, HTTP error responses with "ProblemDetails" data structure are specified as part of the CommonData OpenAPI specification file in Annex A of 3GPP TS 29.571 [5].

Example:

In the example below, the 400, 500 and default error response descriptions defined in 3GPP TS 29.571 [5] are referenced.

```
paths:
  /users:
    get:
      responses:
        '200':
          content:
            application/json
              schema:
                $ref: '#/components/schemas/ExampleGetBody'
        '400':
          $ref: 'TS29571_CommonData.yaml#/components/responses/400'
        '500':
          $ref: 'TS29571_CommonData.yaml#/components/responses/500'
        default:
          $ref: 'TS29571_CommonData.yaml#/components/responses/default'
```

The following definitions provided in Annex A of 3GPP TS 29.571 [5] are used in that example:

```
components:
  responses:
    '400':
      description: Bad request
      content:
        application/problem+json:
          schema:
            $ref: '#/components/schemas/ProblemDetails'
    '500':
      description: Internal Server Error
      content:
        application/problem+json:
          schema:
            $ref: '#/components/schemas/ProblemDetails'
    default:
      description: Generic Error
```

5.3.12 Enumerations

For enumerations, as defined in clause 5.2.4.3, the OpenAPI Specification [4] file shall contain a definition in the components/schemas clause defining a schema with the name of the enumeration as key.

The schema

- shall contain the "anyOf" keyword listing as alternatives:

  1. the "type: string" keyword and the "enum" keyword with a list of all defined values for the enumeration; and

  2. the "type: string" keyword and the "description" keyword with a description stating that the string is only provided for forward compatibility with future extensions but is not used to encode contents defined in the present version of the specification. Future extensions may need to be defined in conjunction with the supported feature mechanism as specified in clause 6.6.2 of 3GPP TS 29.500 [2], and

- should contain a description field, containing the overall meaning and purpose of the enumeration; additionally, this field may contain a list of the defined values of the enumeration together with explanations of those values.

  NOTE: The "enum" keyword restricts the permissible values of the string to the enumerated ones. This can lead to extensibility problems when new values need to be introduced.

Example:

ExampleEnumeration represents xxx, and it is used to indicate yyy or zzz; it complies with the provisions defined in Table 5.3.12-1

Table 5.3.12-1: Enumeration ExampleEnumeration

Enumeration value	Description	Applicability
"ONE"	Value One description	
"TWO"	Value Two description	

The data structure in table 5.3.12-1 is described in an OpenAPI specification file as follows:

```
components:
  schemas:
    ExampleEnumeration:
      anyOf:
      - type: string
        enum:
          - ONE
          - TWO
      - type: string
        description: >
          This string provides forward-compatibility with future
          extensions to the enumeration but is not used to encode
          content defined in the present version of this API.
      description: |
        ExampleEnumeration represents xxx, and it is used to indicate yyy or zzz;
        it complies with the provisions defined in Table 5.3.12-1 of 3GPP TS ab.cde.
        Possible values are:
        - ONE: Value One description
        - TWO: Value Two description
```

5.3.13 Formatting of structured data types in query parameters

Structured data types shall represent JSON objects or arrays.

When used in query parameters of a URI, the following formatting shall be used:

- JSON objects and arrays of JSON objects: they shall be formatted using the JSON syntax, which is specified in OpenAPI Specification [4] by including a "content:" block, and specifying the "application/json" media type, followed by the OpenAPI definition of the object.

  EXAMPLE:

  ```
  - name: plmn-id
    in: query
    content:
      application/json:
        schema:
          type: object
          properties:
            mcc:
              type: string
            mnc:
              type: string
  ```

  This results in the following formatting:

  ```
  .../resource?plmn-id={"mcc":"123","mnc":"456"}
  ```

When an 5GC SBI API omits the inclusion of a "content:" block, a "content:" block specifying the "application/json" media type shall be assumed as default.

- Arrays of simple types: they shall be formatted using the OpenAPI "style" keyword set to "form" and the "explode" keyword set to "false".

  EXAMPLE:

  ```
  - name: service-names
    in: query
    style: form
    explode: false
    schema:
      type: array
  ```

```
            items:
              type: string
```

This results in the following formatting:

```
      .../resource?service-names=service1,service2,service3
```

When an 5GC SBI API omits the inclusion of a "content:" block, a "content:" block specifying the "application/json" media type shall be assumed as default.

As described in 3GPP TS 29.500 [2], clause 5.2.10, when the URI is composed, it shall escape (percent-encode) certain "reserved" characters. When an array of strings is formatted with "explode: false" keyword, the COMMA (U+002C) character is used as separator of the different string values; this implies that this character (the value separator) shall not be escaped, while any other comma used inside the different string values shall be escaped (percent-encoded).

5.3.14   Attribute Presence Conditions

In an OpenAPI Specification [4], presence conditions for attributes in a JSON schema definition shall be expressed by using the "required" keyword, indicating a list (array) of attributes that shall always be present in an object conforming to such schema.

The "required" keyword may be used as part of any of the expressions defined by OpenAPI to combine schemas ("oneOf", "anyOf", "allOf", "not").

   EXAMPLES:

   -   JSON object defining attributes "a" and "b", of type integer, where attribute "a" shall always be present:

```
components:
  schemas:
    ExampleType1:
      type: object
      required: [ a ]
      properties:
        a:
          type: integer
        b:
          type: integer
```

   -   JSON object defining attributes "a" and "b", of type integer, where at least one of them, or both, shall be present:

```
components:
  schemas:
    ExampleType2:
      type: object
      anyOf:
        - required: [ a ]
        - required: [ b ]
      properties:
        a:
          type: integer
        b:
          type: integer
```

   -   JSON object defining attributes "a" and "b", of type integer, where at least one of them shall be present, but not both:

```
components:
  schemas:
    ExampleType3:
      type: object
      oneOf:
        - required: [ a ]
        - required: [ b ]
      properties:
        a:
          type: integer
        b:
          type: integer
```

- JSON object defining attributes "a" and "b", of type integer, where "a" and "b" can be both absent but, if one of them is present, the other shall be absent:

```
components:
  schemas:
    ExampleType4:
      type: object
      not:
        required: [ a, b ]
      properties:
        a:
          type: integer
        b:
          type: integer
```

- JSON object defining attributes "a" and "b", of type integer, where "b" shall be present if "a" takes a given value (e.g., value 1), but may be absent otherwise:

```
components:
  schemas:
    ExampleType5:
      type: object
      properties:
        a:
          type: integer
        b:
          type: integer
      anyOf:
        - not:
            required: [ a ]
            properties:
              a:
                type: integer
                enum: [ 1 ]
        - required: [ b ]
```

- JSON object defining attributes "a" and "b", of type integer, where "b" shall be present if and only if "a" takes a given value (e.g., value 1):

```
components:
  schemas:
    ExampleType6:
      type: object
      properties:
        a:
          type: integer
        b:
          type: integer
      oneOf:
        - required: [ a ]
          properties:
            a:
              type: integer
              enum: [ 1 ]
        - not:
            required: [ b ]
```

- JSON object defining attributes "a", "b", "c" and "d", of type integer, where none of the pairs ("a" and "b") and ("c" and "d") shall be present:

```
components:
  schemas:
    ExampleType7:
      type: object
      properties:
        a:
          type: integer
        b:
          type: integer
        c:
          type: integer
        d:
          type: integer
      allOf:
        - not:
            required: [ a, b ]
```

```
      - not:
          required: [ c, d ]
```

NOTE: Object schema definitions should not have property names in the "required" attribute for which a
          corresponding property definition does not exist.

5.3.15   Usage of the "tags" field

In an OpenAPI specification, all HTTP operations belonging to the same resource should include a "tags" field
containing a same value, briefly describing that resource (e.g. using the name of the resource and its archetype). This
results in all operations being grouped by the User Interface of OpenAPI tools, which helps with readibility of the API
documentation.

EXAMPLE:

```
openapi: 3.0.0
(...)
paths:
  /nf-instances/{nfInstanceID}:
    get:
      summary: Read the profile of a given NF Instance
      operationId: GetNFInstance
      tags:
        - NF Instance ID (Document)
    (...)
    put:
      summary: Register a new NF Instance
      operationId: RegisterNFInstance
      tags:
        - NF Instance ID (Document)
    (...)
    patch:
      summary: Update NF Instance profile
      operationId: UpdateNFInstance
      tags:
        - NF Instance ID (Document)
    (...)
```

5.3.16   Security

As indicated in 3GPP TS 33.501 [22] and 3GPP TS 29.500 [2], the access to an 5GC API may be authorized by means
of the OAuth2 protocol (see IETF RFC 6749 [23]), based on local configuration. 5GC APIs thus need to support the
OAuth2 protocol.

To reflect this, the OpenAPI specification file of an API shall contain:

-   a top-level "security" field, that applies to the overall API unless overridden by a resource/operation-level
    "security" field, listing as alternatives:

    i)   "{}" to indicate that usage of security is optional; and

    ii) the name of the security scheme for oAuth2, including in the array of scopes the name of the API as the only
         scope; and optionally

-   a resource/operation-level "security" field, that applies to a specific operation on a specific resource, and
    overrides the top-level "security" field, listing as alternatives:

    i)   "{}" to indicate that usage of security is optional; and

    ii) the name of the security scheme for oAuth2, including in the array of scopes the name of the API as the only
         scope, and

    iii) the name of the security scheme for oAuth2, including in the array of scopes the name of the API and the
         name of one scope to be used to invoke the specific resource/operation, and optionally

    iv) the name of the security scheme for oAuth2, including in the array of scopes the name of the API and the
         names of two scopes to be used to invoke the specific resource/operation, and

NOTE: the option iv) can be used e.g. to define one scope for a set of specific resources, and a second scope for a specific operation within the set.

- a "securitySchemes" field in the "components" clause defining a security schema for oAuth2 as follows:

   i) to be of type "oauth2"; and

   ii) with a "flows" field containing a "clientCredentials" field that contains:

      1) a "tokenUrl" field pointing to the Access Token Request service provided by the NRF (see 3GPP TS 29.510 [18]); and

      2) a "scopes" field defining all the different scopes applicable to this API, which includes the name of the corresponding API (using the format used within URIs of that API) to be used as the scope required to get access to the overall API, and also including those resource/operation-level scopes to be used as scopes required to invoke a specific operation on a specific resource.

      The naming of the resource/operation-level scopes should consist of the concatenation of the service name, a string or sequence of strings representing a resource name or set of resource names or custom operation, and optionally a string indicating the type of access (e.g. read/modify/create), separated by the ":" (colon) character.

      Such last string component of the resource/operation-level scope, that represents the type of access for a resource, should comply with the following principles:

      - "read": for GET operations for any resource archetype,

      - "create": for POST or PUT operations that result in a creation of new resources from a collection or store resource,

      - "modify": for PUT, PATCH or DELETE operations that result on an update or deletion of a document resource,

      - "invoke": for POST operations that result in the invocation of a custom operation

      - "write": for any operation that results in creation, modification or deletion of resource representations.

Example 1:

```
security:
  - {}
  - oAuth2ClientCredentials:
    - nnrf-nfm

paths:
  /nf-instances:
    get:
      security:
        - {}
        - oAuth2ClientCredentials:
          - nnrf-nfm
        - oAuth2ClientCredentials:
          - nnrf-nfm
          - nnrf-nfm:nf-instances:read
      parameters:
      (...)
      responses:
      (...)

components:
  securitySchemes:
    oAuth2ClientCredentials:
      type: oauth2
      flows:
        clientCredentials:
          tokenUrl: '{nrfApiRoot}/oauth2/token'
          scopes:
            nnrf-nfm: Access to the Nnrf_NFManagement API
            nnrf-nfm:nf-instances:read: Read access to the NF Instances (Collection) resource
```

Example 2:

```
security:
  - {}
  - oAuth2ClientCredentials:
    - nudr-dr

paths:
  /exposure-data/{ueId}/access-and-mobility-data:
    get:
      security:
        - {}
        - oAuth2ClientCredentials:
          - nudr-dr
        - oAuth2ClientCredentials:
          - nudr-dr
          - nudr-dr:exposure-data
        - oAuth2ClientCredentials:
          - nudr-dr
          - nudr-dr:exposure-data
          - nudr-dr:exposure-data:read
      parameters:
      (...)
      responses:
      (...)

components:
  securitySchemes:
    oAuth2ClientCredentials:
      type: oauth2
      flows:
        clientCredentials:
          tokenUrl: '{nrfApiRoot}/oauth2/token'
          scopes:
            nudr-dr: Access to the Nudr_DataRepository API
            nudr-dr:exposure-data: Access to the ExposureData data set
            nudr-dr:exposure-data:read: Access to read ExposureData
```

5.3.17 Reuse of Structured Data Types

As indicated in clause 5.2.4.1, common data types can be defined in 3GPP TS 29.571 [5], in order to avoid the duplication of data type definitions across multiple APIs.

When such data types are of type object, a given API may require that the common data type is extended with additional attributes, on top of those from the original definition in 3GPP TS 29.571 [5].

In such case, the new data type in that API should not re-define entirely the common data type, and instead, it should use appropriate constructs in OpenAPI (i.e. the "allOf" keyword) to indicate that the new data type contains all the attributes form the common data type, by referencing to it, and indicate the additional attributes needed in the new data type.

EXAMPLE:

The ProblemDetails data type is defined in 3GPP TS 29.571 [5] as:

```
ProblemDetails:
  type: object
  properties:
    type:
      $ref: '#/components/schemas/Uri'
    title:
      type: string
    status:
      type: integer
    detail:
      type: string
    instance:
      $ref: '#/components/schemas/Uri'
    cause:
      type: string
    invalidParams:
      type: array
      items:
```

```
        $ref: '#/components/schemas/InvalidParam'
      minItems: 1
```

Then, if a given API needs to extend ProblemDetails with an additional attribute, e.g. "action" of type string, it should define, e.g. an ExtendedProblemDetails data type as follows:

```
ExtendedProblemDetails:
  allOf:
    - $ref: 'TS29571_CommonData.yaml#/components/schemas/ProblemDetails'
    - $ref: '#/components/schemas/ProblemDetailsExtension'

ProblemDetailsExtension:
  type: object
  properties:
    action:
      type: string
```

5.3.18   Operation identifiers

Service operations defined in an OpenAPI specification file should be assigned an Operation ID.

EXAMPLE:

```
get:
  operationId: ReadInfo
  summary: Read Information.
  tags:
    - Information (Document)
  parameters:
    - name: infoType
      in: path
      description: Requested information Type
      required: true
      schema:
        type: string
```
(…)

5.3.19   Formatting of description fields

As indicated in OpenAPI Specification [4], "description" fields support the CommonMark Markdown formatting (see https://spec.commonmark.org/0.27).

In the 3GPP APIs, a description field may consist of multiple lines of YAML text; in order to have a proper output formatting (i.e., to be correctly rendered by tooling producing, e.g., HTML-based content), they should be documented as follows:

- If line breaks need to be kept in the output formatting, this shall be indicated by two trailing white spaces (see https://spec.commonmark.org/0.27/#hard-line-breaks). Those are represented in the example below as gray-shaded non-break spaces for illustration purposes, but the YAML file shall contain two normal white spaces:

```
description: |
  Text in the first line.▨
  More text in the second line.▨
  Additional text in the third line.
```

- If the text in the different lines should be formatted as a bulleted HTML list, each line may be prepended by the hyphen "-" character, as follows:

```
description: |
  Text in the first line:
  - List item 1
  - List item 2
```

- If the text should be displayed in the output formatting as a single line, the YAML file may be formatted using one of the following options:

  - All text in a single YAML line, following the "description" keyword, such as:

```
description: This is a long description text that is formatted as a single YAML line, which
is correct, but may not be optimal when the YAML file is displayed on a text editor
configured to not automatically wrap long lines.
```

- The "description" keyword is followed by the ">" character and the text is placed in different lines (this option should be the preferred approach, rather than having extremely long lines in the YAML file), such as:

```
description: >
  This text is the first part of a single line,
  followed by this second line,
  where all these 3 lines will be displayed in the same line in the output formatting.
```

6 Requirements for secure API design

6.1 Introduction

This clause contains a list of security requirements for API design provided by SA3.

6.2 General

The following requirements are intended as general guidance for 3GPP Stage 3 work in order to specify secure protocols and APIs. As such, these guidelines are independent of the specific technology and shall be followed at all times.

- The valid format and range of values (when applicable) for each IE shall be defined unambiguously.

NOTE 1: Explicitly defining format and range of values not only helps to improve the security of a certain implementation, but also allows for reliable interoperability between different protocol implementations. Example: Defining a "lowercase string variable of length 10 and range [a..z]" is much more explicit that just defining a "string of length 10". There are known vulnerabilities such as a denial of service (resulting in the parser converting from a string representing an integer – an attacker can pass in an arbitrarily large integer and trigger an unhandled exception) and such leading to a heap corruption and crash (proof-of-concept available), or potentially remote code execution (no proof-of-concept known). Unicode literals also require special treatment when doing string comparisons to ensure that equivalent strings return true when compared.

- For each message the number of leaf IEs shall not exceed 2048K. If a leaf IE is an array of a simple data type, then the whole array shall count as one leaf. If a leaf IE is a data structure or an array of data structures, then it shall be considered a branch, i.e. it shall not be counted as a leaf. The data structure's (branch) attributes determine the number of leaves. For instance, a data structure with e.g. three attributes will count as three leaves.

- The maximum size of the JSON body of any HTTP request/response shall not exceed 16 million octets before compression is applied, if any.

NOTE 2: APIs need to be designed taking care to avoid a too large HTTP content size for performance reasons.

- The maximum nesting depth of leaves shall not exceed 32. If a leaf IE is an array of a simple data type, then the whole array shall be considered as the first level of nesting. If a leaf IE is a data structure or an array of data structures, then it shall be considered a branch and the first level of nesting. The data structure's attributes (leaves) shall be considered as the second level of nesting. For instance, a data structure with e.g. one attribute-A, which is also a data structure with e.g. one attribute-B, then attribute-B will make the third level of nesting.

NOTE 3: There are resource exhaustion attacks on JSON parsers. Defined maximum numbers of IEs, sizes and nesting depths allow implementations to know an upper bound of required resources. It also allows validation of incoming messages. Recursively processing nested objects leads to stack exhaustion and a denial of service bug.

- For data structures where values are accessible using names (sometimes referred to as keys), e.g. a JSON object, the name shall be unique. The occurrence of the same name (or key) twice within such a structure shall be an error and the message shall be rejected.

NOTE 4: Serialization schemes (e.g. JSON) can leave the handling of repeated names (keys) up to the implementer's discretion. For example, for a repeated name an error can be raised, the pair can be ignored, or the first or last value read can be used, though there is no canonical order in which a parser should treat the data it receives. Failure to adhere to consistent handling rules can lead to vulnerabilities. From a security perspective rejecting objects with repeated names, rather than accepting according to some rule, is the more robust solution, and aids in identification of potentially malicious activity. There are known attacks with specially crafted malicious messages that are designed to confuse implementations of NFs to get fraudulent messages into a PLMN.

6.3 SBA-specific requirements

The following requirements shall be considered for every network function that implements a service-based interface.

- OpenAPI specifications are machine-readable JSON objects and can be used as the basis for re-configuring an NFs action when an API or message structure changes. Therefore, each OpenAPI specifications shall contain all necessary information to correctly and unambiguously parse the contents of the message body.

NOTE: Attacks often exploit corner cases and unspecified behavior in order to exploit a system. Traffic normalization counters this by either dropping traffic that is malformed or by forcing certain information elements to a "normal" value. Typically, this relates to inconsistent fields.

- 3GPP TS 33.501 [22] documents which type of information shall be confidentiality protected on the N32 interface. The fields where these types of information (e.g. SUPI) is contained may have different names. Even if the field names are different, the mechanism specified in clause 5.2.3.3 of 3GPP TS 29.573 [24] shall clearly identify the type of information carried in each IE and which information types shall be confidentiality protected.

Annex A (informative):
TS Skeleton Template

A TS Skeleton Template to be used as a starting point of drafting a 5G System SBI Stage 3 specification is available at the following location:

https://www.3gpp.org/ftp/information/All_Templates/29.xxx-SBI-Stage3-Template.zip

Annex B (informative):
Backward Incompatible Changes

This Annex provides information about the changes in the API that are considered as backwards compatible and those that are considered as backwards incompatible. This list is to be considered informative and it may be expanded in future releases, when necessary.

Backward compatible changes are additions or changes in the API that do not break the existing Service Consumer behaviour. Examples of backward compatible changes include:

- Adding a new, optional child resource/URI;

- Supporting a new HTTP method;

- Adding new elements to a resource representation;

- Changing the order of fields in a resource representation;

- Addition of a new status code:

NOTE 1: When a NF / NF Service receives a HTTP status code that it cannot recognize it will treat it as the corresponding x00 status code as specified in clause 5.2.7.3 of 3GPP TS 29.500 [2].

- Corrections of obvious errors in an OpenAPI file required to enable a correct parsing of the file such as misspelled references;

- Corrections that only relate to smaller and optional parts of the functionality (e.g. a supported feature, see 3GPP TS 29.500 [2] clause 6.6.2), even if the changes are backward incompatible with respect to that part of the functionality; and

NOTE 2: It is recommended to only apply corrections which are also backward compatible with respect to such smaller and optional parts of the functionality. If this is not possible a new supported feature can be introduced to enable a negotiation of the support of the correction, and the old corresponding supported feature can be marked as "withdrawn" in the table defining the supported features of an API.

- Backward-compatible changes related to the semantics (i.e. functional behaviour) specified for an API.

Changes in the API that do not result in any loss of existing functionality (i.e. functionality that works fine if both consumer and provider do not support the change) if only consumer or only provider implements the change can be considered as backwards compatible corrections or additions.

Backward incompatible changes are additions or changes in the API that break the existing Service Consumer behaviour. Here is a list of backward incompatible changes that shall require incrementing the 1st field (MAJOR) of the API version number unless they only relate to smaller and optional parts of the functionality (see above):

- Removing a resource/URI:

- Removing support for an HTTP method;

- Renaming a field in a resource representation;

- Adding mandatory parameters to a resource URI or resource representation;

- Attribute data type changes;

- Cardinality changes (NOTE 3); and

NOTE 3: Whether attribute cardinality changes are backward compatible depend on the type of change. Examples of non-backward compatibility changes include decreasing the upper bound of a cardinality range for attributes sent by the NF service consumer, changing the meaning of the default behavior associated to the absence of an attribute of cardinality 0..N, etc.

- Backward incompatible changes related to the semantics (i.e. functional behaviour) specified for an API.

Changes in the API that result in loss of existing functionality (i.e. functionality that works fine if both consumer and provider do not support the change) if only consumer or only provider implements the change can be considered as backwards incompatible modifications.

When a change although being categorised as backwards compatible correction or addition results in interoperability issues, it is expected that the issue will be resolved by implementing the change at both consumer and provider.

Annex C (Informative):
Resource modelling

C.0   General

When designing an API, one shall first think of defining the set of resources consumed. Resources represent objects that are modified by standard HTTP methods and that can be modelled with one of 4 archetypes detailed below. Resource archetypes help API designers to structure the resources. In this process the designer should refer to the appropriate archetype when the resource definition perfectly matches the archetype one. Referring to an archetype immediately defines what operations and HTTP methods are supported by the resource.

The archetypes provided hereafter don't preclude the existence of resources of different types.

C.1   Document

The document archetype is the conceptual base archetype of the other ones. Any resource that is not identified with one of the other resource archetypes is a document.

A document may have child resources that represent its specific subordinate concepts.

The archetype does not place any restriction on HTTP methods when acting on a document.

Only CRUD operations are performed directly on a document resource, i.e. by sending an HTTP request to the URI of that resource. Custom methods are not performed directly on the resource, but by sending an HTTP request to a URI that is associated by a convention with the URI of the resource.

C.2   Collection

The collection archetype can be used to model a resource that serves as a directory of resources. A collection is NF Service Provider-managed so the NF Service Provider decides the URIs of each resource that is created in the collection.

> NOTE: Even though a collection resource typically contains child resources, it is allowed that a particular collection resource does not contain any child resource at a particular point in time ("empty collection").

The Create and Read operations are performed on a collection directly.

More specifically:

- A collection child resource is created by sending a POST with the collection URI if accepted by the collection;

- A collection is read by sending a GET with the collection URI;

- The PUT and PATCH methods with the collection URI are not allowed;

- The DELETE method with the collection URI is only allowed if the collection resource has been created dynamically based on a request from the NF Service Consumer.

- The authorized operations on a collection child resource depend on that resource's archetype.

C.3   Store

The store archetype can also be used to model a resource that serves as a directory of resources but a store is NF Service Consumer-managed. The NF Service Consumer solely decides what resource shall be added to / deleted from a store. The NF Service Consumer decides what the URI of the added resource is.

NOTE: Even though a store resource typically contains child resources, it is allowed that a particular store resource does not contain any child resource at a particular point in time ("empty store").

The Read operation is performed on a store directly, and the Create operation is performed on store child resources.

More specifically:

- A store child resource is created by sending a PUT with the URI of the child resource to be created.

- A store is read by sending a GET with the store URI;

- The POST, PUT and PATCH methods with the store URI are not allowed;

- The DELETE method with the store URI is only allowed if the store resource has been created dynamically based on a request from the NF Service Consumer.

- Apart from Create (PUT), the authorized operations on a store child resource depend on that resource's archetype.

C.4 Custom operation

The custom operation archetype can be used to model an unsafe and non-idempotent operation that is not a Create on a collection.

A custom operation does not operate directly on the resource that would be identified by the custom operation URI. Instead, when the custom operation is associated with a resource, the operation is performed on this associated resource. For instance, a custom operation may modify the associated resource in a special way. This associated resource is identified by stripping the suffix string "/<custOpName>" from the custom operation URI template in clause 4.4.2.

When the custom operation is not associated with any resource but with the service, it acts as an executable function with input parameters and returns the result of the executed function in the response body, not modifying any resource.

POST is the only method allowed with a custom operation URI.

The semantic of the custom operation is encoded in the last segment of the URI template in clause 4.4.2: /{custOpName}.

Annex D (informative): Example of an OpenAPI specification file for Patch

As described in clause 4.6.1.1.3.2, the bodies of HTTP PATCH requests will either use a "JSON Merge Patch" encoding as defined in IETF RFC 7396 [7], a "JSON Patch" encoding as defined IETF RFC 6902 [8], or an HTTP multipart message with the multipart/mixed content-type as described in IETF RFC 2046 [29]. This annex provides an example OpenAPI Specification [4] allowing both encodings.

NOTE: Both encoding possibilities are shown in this example for illustrative purposes. However, only a single of the above encodings will be specified for each resource where the PATCH method is supported unless backward compatibility considerations necessitate the support of both encodings.

```
openapi: 3.0.0

info:
  version: "1.0.0"
  title: PATCH Example

paths:
  /inventory:
    post:
      summary: adds an inventory item
      operationId: addInventory
      description: Adds an item to the system
      responses:
        '201':
          description: item created
        '400':
          description: 'invalid input, object invalid'
        '409':
          description: an existing item already exists
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/InventoryItem'
        description: Inventory item to add

  /inventory/{id}:
    get:
      summary: read inventory item
      parameters:
        - name: id
          in: path
          required: true
          schema:
            type: integer
      responses:
        '200':
          description: search results matching criteria
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/InventoryItem'
        '400':
          description: bad input parameter
    patch:
      summary: patch inventory item
      parameters:
        - name: id
          in: path
          required: true
          schema:
            type: integer
      requestBody:
        required: true
        content:
          application/json-patch+json:
            schema:
              $ref: '#/components/schemas/PatchInventoryItem'
          application/merge-patch+json:
            schema:
              $ref: '#/components/schemas/MergePatchInventoryItem'
```

```
              multipart/mixed:
                schema:
                  $ref: '#/components/schemas/MultipartPatch'
                encoding:
                  patch: # The patch part shall be the first part
                    contentType: application/json-patch+json
                    headers:
                      Content-Id:
                        schema:
                          type: string
                        required: true
                  binaryBlocks: # 0 or more block parts may follow the patch part
                    contentType: '*/*' # Block part can be of any type
                    headers:
                      Content-Id: # Block identifier is defined by the Content-Id header.
                        schema:
                          type: string
                        required: true
                      Content-Transfer-Encoding:
                        schema:
                          type: string
                        required: true

      responses:
        '200':
          description: Patch was succesfull and updated Inventory Item is returned.
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/InventoryItem'
        '204':
          description: Patch was succesfull
        '400':
          description: bad input parameter

components:

  schemas:
    InventoryItem:
      type: object
      required:
        - name
        - manufacturer
      properties:
        id:
          type: integer
        name:
          type: string
        manufacturer:
          $ref: '#/components/schemas/Manufacturer'
        customers:
          type: array
          items:
            type: string

    Manufacturer:
      type: object
      description: Represents the manufacturer.
      required:
        - name
      properties:
        name:
          type: string
        homePage:
          type: string
          format: url
        phone:
          type: string

    ManufacturerRm:
      type: object
      description: >
        This data type is defined in the same way as the "Manufacturer" data type,
        but with the OpenAPI "nullable: true" property.
      required:
        - name
      properties:
        name:
```

```yaml
            type: string
        homePage:
          type: string
          format: url
        phone:
          type: string
      nullable: true


    PatchInventoryItem:
      type: array
      description: A JSON PATCH body schema to Patch selected parts of an Inventory Item
      items:
        anyOf:
          - oneOf:
            - type: object
              description: Modifies the URL of a Manufacturer
              properties:
                op:
                  type: string
                  enum:
                    - "add"
                    - "remove"
                    - "replace"
                path:
                  type: string
                  pattern: '^\/manufacturer\/homePage$'
                value:
                  type: string
                  format: url
              required:
                - "op"
                - "path"
            - type: object
              description: Modifies a Manufacturer
              properties:
                op:
                  type: string
                  enum:
                    - "replace"
                path:
                  type: string
                  pattern: '^\/manufacturer$'
                value:
                  $ref: '#/components/schemas/Manufacturer'
              required:
                - "op"
                - "path"
                - "value"
            - type: object
              description: Modifies a Customer
              properties:
                op:
                  type: string
                  enum:
                    - "add"
                    - "remove"
                    - "replace"
                path:
                  type: string
                  pattern: '^\/customers\/(-|\d+)$'
                value:
                  type: string
              required:
                - "op"
                - "path"
          - type: object
            description: Open Alternative
      minItems: 1

    MergePatchInventoryItem:
      description: A JSON Merge PATCH body schema to Patch selected parts of an Inventory Item
      type: object
      properties:
        manufacturer:
          $ref: '#/components/schemas/ManufacturerRm'
        customers:
          type: array
```

```
              description: Allows to replace the entire array, but not to modify individual elements.
              items:
                type: string


    BinaryBlock:
      description: A Block can be of any type
      example: >-
        "QmxvY2sgY29udGVudA=="


    MultipartPatch:
      description: Definition of a Record Patch
      type: object
      properties:
        patch:
          # json patch as defined in RFC 6902
          type: array
          items:
            $ref: 'TS29571_CommonData.yaml#/components/schemas/PatchItem'
          minItems: 1
        blocks:
          # List of multipart data
          type: array
          description: list of opaque Block's being reffered to by PatchItems
          items:
            $ref: '#/components/schemas/BinaryBlock'
          minItems: 1
      required:
        - patch
      example: >-
        {"patch": [{ "op": "add", "path": "/firstBodyPart/callbackReference", "value":
"https://example.com" }, { "op": "copy", "from": "/binaryBlocks/blockID1", "to":
"/Binaryblocks/blockID2" }, { "op": "add", "path": "/binaryBlocks/blockID1", "value": "addedBlock1"
}], "blocks": [{"Content-Id": "addedBlock1", "Content-Type": "text/plain", "content":
"QmxvY2sgY29udGVudA=="}]}
```

Annex E (Informative): Considerations for handling of JSON arrays

This Annex provides guidelines on the use of JSON arrays on the SBI APIs of the 5GC.

As described in clause 5.2.4, the data model of an API definition in a 5GC API consists of any of the different data types supported by OpenAPI, corresponding to the different data structures found in the JSON data format. One of these structures is the "array", representing a set of ordered values.

It should be noted that, while JSON allows that the value of the elements of an array may be of different types, in OpenAPI Specification [4]this is further restricted, and all elements of an array shall be of the same type.

Also, it is important to note that the JSON format itself, as specified in IETF RFC 8259 [3], does not define any syntax to refer to specific array elements.

However, there are certain conventions to specify mechanism to refer to array elements, e.g. based on the position a given element has in the array.

In particular, the JSON Pointer syntax defines a string syntax for identifying a specific value within a JSON document. This syntax consists of a number of tokens separated by the "/" character; in order to refer to a specific element in an array, then token shall contain an unsigned decimal value, indicating the zero-based index of the element in the array.

> EXAMPLE:
>
> JSON document:

```
{
  "attr1": 0,
  "attr2": true,
  "attr3": [ 1, 2, 3 ]
}
```

> JSON Pointer expression "/attr3/0" evaluated on such JSON document: 1

There are several scenarios, frequently employed in the 5GC APIs, that make use of the JSON Pointer mechanism to refer to specific elements in an array:

- Update of resources using the PATCH method, as described in clause 4.6.1.1.3.2. If the syntax used in the PATCH request payload is based on the "JSON Patch" format, then the JSON Pointer mechanism is used to specify patch operations applied to specific array elements.

- Notifications of events (such as data changes), as described in clause 4.6.2.3, using as notification payload the notation defined in "NotifyItem" / "ChangeItem" data types defined in 3GPP TS 29.571 [5]. This notation is similar to "JSON Patch", so it also makes use of the JSON Pointer syntax to refer to specific array elements.

- Explicit usage of attributes containing JSON Pointer expressions in request or response payloads.

In these scenarios, it is critical that any JSON Pointer expression is applied by both client and server on the exact same array representation, since otherwise the indexes may vary, and the JSON Pointer will give unexpected results.

A typical scenario that may create issues could be as follows:

1. NF Service Consumer sends a first GET request towards a NF Service Producer, including certain query parameters in the HTTP request, that result into retrieving a resource representation that contains a subset of the data that the NF Service Producer holds under such resource. When such subset refers to returning just some of the elements of an array, rather than the entire array, then the content of the array will differ between consumer and producer.

2a. The NF Service Consumer sends a subsequent PATCH request towards the NF Service Producer, with the intention to modify a given element of the array (specified by the array index, per the JSON Pointer syntax). This results into the server modifying a wrong element in the array, given that the NF Service Producer contains a different array.

2b. The NF Service Consumer may subscribe to be notified by the NF Service Producer when a given resource representation has changed. When the NF Service Producer detects such a change, it sends a notification that

may include a reference to an array index, which may be different than the array index kept by the NF Service Producer.

2c. The NF Service Consumer is a stateless NF Service Consumer that has no local representation of the array that subscribes to be notified by the NF Service Producer when a given resource representation has changed.

Another scenario that may lead to incorrect array updates is:

1. Two different NF Service Consumer sends a GET request towards a NF Service Producer to retrieve the representation of a certain resource.

2. A first NF Service Consumer NF1 sends a PATCH request towards the NF Service Producer, with the intention to delete a given element of the array (specified by the array index, per the JSON Pointer syntax). This results in some of the array indexes being changed (of those elements placed after the deleted element)

3. A second NF Service Consumer NF2, sends another PATCH request towards the NF Service Producer, to modify any of the elements of the array whose index was affected by the previous operation done by NF1. This results in modifying unintendedly the wrong array element.

The design of 5GC SBI APIs should take into account these scenarios and provide mechanisms to avoid unintended modifications of array elements, when they are referred by their position index in the array.

To achieve these, both NF Service Consumer and Producer (taking the role of HTTP client and server) should ensure that any resource update takes place on a known and current resource representation, based on the content of ETag values sent along with resource representations by the resource owner.

If an NF Service Consumer needs to cache a resource representation received from an NF Service Producer (i.e. the JSON information received in an HTTP response message), it shall use the exact same representation of arrays as received from the service producer.

When sending notifications or modifications whose semantics is based on the JSON Pointer syntax, the sending NF shall use the exact same representation of arrays as previously signalled in a previous interaction with the receiver of such notifications/modifications.Further, when sending notifications whose semantics is based on the JSON Pointer syntax, if the NF Service Consumer is a stateless NF Service Consumer and thus has no local representation of the array, the NF Producer shall use the complete replacement array representation as the NF Service Consumer is otherwise required to request the complete data from the NF Service Producer with a GET request.

Annex F (informative):
Change history

Change history							
Date	Meeting	TDoc	CR	Rev	Cat	Subject/Comment	New version
2017-10	CT4#80	C4-175250				TS skeleton	0.1.0
2017-10	CT4#80	C4-175358 C4-175252 C4-175253 C4-175254 C4-175255 C4-175331 C4-175332 C4-175333 C4-175334 C4-175359 C4-175327 C4-175328 C4-175360 C4-175330 C4-175336 C4-175337				Inclusion of pCRs agreed at CT4#80	0.2.0
2017-12	CT4#81	C4-176414 C4-176372 C4-176447 C4-176415 C4-176416 C4-176417 C4-176418 C4-176250 C4-176419 C4-176422				Inclusion of pCRs agreed at CT4#81	0.3.0
2018-01	CT4#82	C4-181179 C4-181384				Inclusion of pCRs agreed at CT4#82	0.4.0
2018-03	CT4#83	C4-182396 C4-182397 C4-182394 C4-182395 C4-182399 C4-182261 C4-182184 C4-182330 C4-182398 C4-182332				Inclusion of pCRs agreed at CT4#83	0.5.0
2018-03	CT#79	CP-180029				Presented for information	1.0.0
2018-04	CT4#84	C4-183238 C4-183288 C4-183289 C4-183291 C4-183292 C4-183385 C4-183387 C4-183388 C4-183477 C4-183478				Inclusion of pCRs agreed at CT4#84	1.1.0
2018-05						29.xxx-SBI-Stage3 Template added in zip-file.	1.1.1
2018-05	CT4#85	C4-184492 C4-184493 C4-184494 C4-184495 C4-184496 C4-184544 C4-184614 C4-184503 C4-184497 C4-184592				Inclusion of pCRs agreed at CT4#85	1.2.0
2018-06	CT#80	CP-181099				Presented for approval	2.0.0
2018-06	CT#80					Approved in CT#80.	15.0.0
2018-07						TS template added in zip-file	15.0.1
2018-09	CT#81	CP-182054	0001	4	B	Security requirements for API design	15.1.0
2018-09	CT#81	CP-182054	0002		F	Example URIs in figures	15.1.0
2018-09	CT#81	CP-182054	0003	2	F	Clarification on the use of API version number	15.1.0
2018-09	CT#81	CP-182054	0004	2	F	External Docs Clause in OpenAPI file	15.1.0
2018-09	CT#81	CP-182054	0006	1	B	JSON Structures in Query Parameters	15.1.0
2018-09	CT#81	CP-182054	0008	1	F	Servers Selection in OpenAPI	15.1.0
2018-09	CT#81	CP-182054	0009		F	Query Parameter	15.1.0
2018-09	CT#81	CP-182054	0010		F	yaml file names	15.1.0

2018-09	CT#81	CP-182054	0011		F	OpenAPI servers field	15.1.0
2018-09	CT#81	CP-182054	0012	1	F	Query parameter	15.1.0
2018-09	CT#81	CP-182054	0013	1	F	presence condition and cardinality	15.1.0
2018-09	CT#81	CP-182054	0014	1	F	Clarification on Naming Conventions and Digits	15.1.0
2018-09	CT#81	CP-182054	0015	1	F	URIs of created resources	15.1.0
2018-09	CT#81	CP-182054	0016	1	F	Custom operation in resource structure presentation	15.1.0
2018-12	CT#82	CP-183012	0018	2	F	Attribute Presence Conditions	15.2.0
2018-12	CT#82	CP-183012	0019	1	F	Version addressed by references within OpenAPI files	15.2.0
2018-12	CT#82	CP-183012	0020	2	F	Resolve Editor's Note	15.2.0
2018-12	CT#82	CP-183012	0021		F	Attribute with Any Type	15.2.0
2018-12	CT#82	CP-183012	0022		F	Incorrect resource URI structure presentation	15.2.0
2018-12	CT#82	CP-183012	0023	1	F	Storage of OpenAPI files within a central directory	15.2.0
2018-12	CT#82	CP-183012	0025	3	F	Complex query expression	15.2.0
2018-12	CT#82	CP-183012	0031	1	F	Correction and Clarification on Security Requirements	15.2.0
2018-12	CT#82	CP-183012	0032	2	R	Subscription Lifetime for Subscribe / Notify operations	15.2.0
2018-12	CT#82	CP-183012	0033		F	Usage of the "tags" field in OpenAPI	15.2.0
2018-12	CT#82	CP-183012	0035	2	F	Corrections to API versioning	15.2.0
2018-12	CT#82	CP-183012	0036	1	F	Security in Open API specification files	15.2.0
2018-12	CT#82	CP-183012	0037	1	F	Custom Operations	15.2.0
2019-03	CT#83	CP-190017	0039	2	F	Address Editor's Note on implicit subscriptions	15.3.0
2019-03	CT#83	CP-190017	0040		F	Address Editor's Note on partial representation in POST response	15.3.0
2019-03	CT#83	CP-190017	0041	1	F	Maximum HTTP payload size	15.3.0
2019-03	CT#83	CP-190017	0042	1	F	HTTP Scheme	15.3.0
2019-03	CT#83	CP-190017	0043	2	F	Resolve Editor's Notes	15.3.0
2019-03	CT#83	CP-190017	0044		F	Correction to Minor Field Increment Rules in API Versioning	15.3.0
2019-03	CT#83	CP-190017	0045	1	F	Handling of unknown attributes during resource creation and modification	15.3.0
2019-03	CT#83	CP-190017	0046	1	F	Correct use of "OpenAPI" name	15.3.0
2019-03	CT#83	CP-190017	0047	1	F	Resolution of Editor´s Notes in Annex C	15.3.0
2019-03	CT#83	CP-190017	0048	2	F	IANA registration of "3gppHal+json" media type	15.3.0
2019-03	CT#83	CP-190017	0049	1	F	Addition of applicability column to query parameters table	15.3.0
2019-03	CT#83	CP-190017	0050	1	F	Removing multiple redundant appearances of major version number in TS template to ease update of that number	15.3.0
2019-03	CT#83	CP-190017	0051	2	F	Storage of OpenAPI specification files	15.3.0
2019-03	CT#83	CP-190017	0052	-	F	Use of Relative URI in Location Header	15.3.0
2019-06	CT#84	CP-191028	0053	2	F	Criteria for non-backward compatible changes	15.4.0
2019-06	CT#84	CP-191028	0056	1	F	Reuse of Structured Data Types	15.4.0
2019-06	CT#84	CP-191028	0057	1	F	Delete editor's note on complex query	15.4.0
2019-06	CT#84	CP-191028	0059	-	F	Correction on Notifications	15.4.0
2019-06	CT#84	CP-191028	0060	1	F	Copyright Note in YAML files	15.4.0
2019-06	CT#84	CP-191028	0055	2	F	Precedence of OpenAPI file	15.4.0
2019-06	CT#84	CP-191028	0054	4	F	Withdrawing API versions	15.4.0
2019-06	CT#84	CP-191057	0061	2	F	Updates to TS Template for SBI	16.0.0
2019-09	CT#85	CP-192101	0065	1	F	Externaldoc version number change	16.1.0
2019-12	CT#86	CP-193025	0069	2	A	Fourth field of API versions	16.2.0
2019-12	CT#86	CP-193036	0070	1	F	Add Corresponding API descriptions in clause 5.1	16.2.0
2019-12	CT#86	CP-193063	0066	2	B	HAL-forms media type	16.2.0
2019-12	CT#86	CP-193063	0067	1	F	Extending ProblemDetails for Backward Compatibility	16.2.0
2020-03	CT#87e	CP-200054	0072	1	F	Storage of YAML files in ETSI Forge	16.3.0
2020-03	CT#87e	CP-200039	0073	1	F	New tables in 29.501 template	16.3.0
2020-06	CT#88e	CP-201034	0074	2	F	Clarifications to callback URI	16.4.0
2020-06	CT#88e	CP-201034	0075	1	B	Resource-Level Authorization	16.4.0
2020-06	CT#88e	CP-201034	0077	2	F	Delimiters - URI and SBI specific	16.4.0
2020-06	CT#88e	CP-201034	0078	2	F	URI types	16.4.0
2020-06	CT#88e	CP-201034	0080	-	F	Correct the Resource Tree Example	16.4.0
2020-06	CT#88e	CP-201034	0081	1	F	Requirements for secure API design	16.4.0
2020-06	CT#88e	CP-201034	0079	1	F	API URI correction and template udpate	16.4.0
2020-06	CT#88e	CP-201034	0082	1	F	Maximum JSON payload size	16.4.0
2020-06	CT#88e	CP-201071	0076	-	F	Correction of the method used for Notification	16.4.0
2020-06	CT#88e	CP-201071	0083	1	F	Missing abbreviations	16.4.0
2020-09	CT#89e	CP-202043	0093	-	A	Storage of YAML files in 3GPP Forge	16.5.0
2020-09	CT#89e	CP-202109	0085	2	D	Editorial fixes	16.5.0
2020-09	CT#89e	CP-202110	0086	2	F	Handling of JSON Arrays	16.5.0
2020-09	CT#89e	CP-202110	0087	-	F	Naming requirements for resource/operation-level scopes	16.5.0
2020-09	CT#89e	CP-202110	0088	1	F	Data types with string type	16.5.0
2020-09	CT#89e	CP-202110	0089	-	F	Content of Location header when TLS is used	16.5.0
2020-09	CT#89e	CP-202110	0090	-	F	Miscellaneous corrections	16.5.0
2020-09	CT#89e	CP-202110	0091	1	F	API URI correction and template update	16.5.0
2020-12	CT#90e	CP-203064	0094	-	F	API Versioning System based on Semantic Versioning	17.0.0
2020-12	CT#90e	CP-203064	0095	-	F	OpenAPI Reference	17.0.0
2020-12	CT#90e	CP-203064	0096	-	F	OpenAPI "description" fields in data type definitions	17.0.0

2021-03	CT#91e	CP-210027	0097	-	D	Use of inclusive terminology	17.1.0
2021-03	CT#91e	CP-210021	0098	-	F	Guidance on Error Response design for Indirect Communication	17.1.0
2021-03	CT#91e	CP-210021	0099	1	B	SBI TS Template Update	17.1.0
2021-03	CT#91e	CP-210021	0101	-	D	Editorial Corrections	17.1.0
2021-03	CT#91e	CP-210021	0102	-	F	Withdrawing API versions	17.1.0
2021-06	CT#92e	CP-211065	0105	-	A	Nested cardinality	17.2.0
2021-06	CT#92e	CP-211028	0103	1	F	Change Notification of Array to Stateless NF Consumer	17.2.0
2021-06	CT#92e	CP-211028	0103	1	B	Additional guidelines for OpenAPI specification files	17.2.0
2021-09	CT#93e	CP-212026	0107	1	F	Compatibility Clarification	17.3.0
2021-09	CT#93e	CP-212026	0108	2	F	Clarification to API name guidance in the SBI template	17.3.0
2021-09	CT#93e	CP-212026	0109	2	F	Corrections to the API URI and relative URI below root concepts	17.3.1
2021-12	CT#94e	CP-213085	0110	1	F	Miscellaneous clarifications	17.4.0
2021-12	CT#94e	CP-213087	0111	5	F	Resource URI structure in the Guidelines for Services Definition	17.4.0
2021-12	CT#94e	CP-213086	0112	2	F	Resource URI structure in the SBI skeleton	17.4.0
2021-12	CT#94e	CP-213086	0114	-	F	Subscription update with PUT request	17.4.0
2022-03	CT#95e	CP-220023	0115	1	F	Adding examples to the Resource URI structure clause	17.5.0
2022-03	CT#95e	CP-220023	0116	1	F	Handling Description field along with the '$ref' keyword	17.5.0
2022-03	CT#95e	CP-220023	0117	1	F	Aligning SBI-Stage3-Template with TS 29.501	17.5.0
2022-03	CT#95e	CP-220023	0118	-	F	JSON schema example with a combination of restrictions	17.5.0
2022-03	CT#95e	CP-220024	0122	-	F	URL Encoding	17.5.0
2022-03	CT#95e	CP-220024	0120	1	D	Miscellaneous corrections to the 5GC SBI TS Skeleton	17.5.0
2022-03	CT#95e	CP-220025	0119	1	F	Formatting of Description Fields	17.5.0
2022-03	CT#95e	CP-220025	0121	1	F	Notification Clarification	17.5.0
2022-06	CT#96	CP-221027	0123	1	F	SBI stage 3 template update - OpenAPI	17.6.0
2022-06	CT#96	CP-221027	0124	1	F	SBI stage 3 template update - Data models	17.6.0
2022-06	CT#96	CP-221027	0125	-	F	Remove the apiVersion placeholder from the resource URI variables table	17.6.0
2022-06	CT#96	CP-221028	0127	-	F	oAuth2 clarification	17.6.0
2022-06	CT#96	CP-221029	0126	1	F	Default logical relationship between query parameters	17.6.0
2022-09	CT#97e	CP-222070	0129	1	A	Essential Correction for Location Header in 201 Response	17.7.0
2022-09	CT#97e	CP-222026	0130	1	F	Limits for number and length of query parameters	17.7.0
2022-12	CT#98e	CP-223027	0132	-	F	Standalone Notifications	18.0.0
2022-12	CT#98e	CP-223029	0131	1	F	Description fields in enumerations	18.0.0
2023-03	CT#99	CP-230090	0137	2	F	Maximum number of leaves	18.1.0
2023-03	CT#99	CP-230029	0134	1	D	Editorial modifications to the rules for incrementing API version	18.1.0
2023-03	CT#99	CP-230029	0138	1	F	Guidelines for OAuth2 scopes definition	18.1.0
2023-03	CT#99	CP-230029	0139	-	F	Correction of the NO-$REF SIBLINGS error for nullable attributes within PATCH method	18.1.0
2023-06	CT#100	CP-231026	0140	1	B	Adding Redirection clause to SBI template	18.2.0
2023-06	CT#100	CP-231027	0133	2	F	Rules for incrementing API version field values	18.2.0
2023-09	CT#101	CP-232033	0142	2	B	Prefix in Callback URIs	18.3.0
2023-09	CT#101	CP-232033	0143	-	B	Documentation of 3GPP Custom HTTP Headers	18.3.0
2023-09	CT#101	CP-232033	0144	1	F	Correct SBI stage 3 template for custom operations without resources	18.3.0
2023-09	CT#101	CP-232033	0145	-	B	Uptade of the template to add ABNF	18.3.0
2023-12	CT#102	CP-233028	0148	1	F	ProblemDetails RFC 7807 obsoleted by 9457	18.4.0
2023-12	CT#102	CP-233028	0149	1	F	Update of SBI TS skeleton template	18.4.0
2023-12	CT#102	CP-233030	0151	-	F	Case-insensitive hexadecimal encoding	18.4.0
2023-12	CT#102	CP-233056	0150	1	B	Optional usage of 3GPP Forge as normative storage of OpenAPI files	18.4.0
2023-12	CT#102	CP-233062	0147	-	A	wrong CR implementation	18.4.0
2024-06	CT#104	CP-241028	0154	1	F	Query Parameter clarification	18.5.0
2024-06	CT#104	CP-241028	0155	1	F	HTTP Multipart message support in PATCH Request	18.5.0
2024-09	CT#105	CP-242039	0162	-	F	HTTP RFCs obsoleted by IETF RFC 9110 and 9113	18.6.0
2024-12	CT#106	CP-243050	0175	-	A	Query parameter clarification	18.7.0
2024-12	CT#106	CP-243022	0164	1	F	Restore deleted clause	18.7.0

History

Document history		
V18.4.0	May 2024	Publication
V18.5.0	July 2024	Publication
V18.6.0	September 2024	Publication
V18.7.0	January 2025	Publication