ETSI TS 132 160 V18.5.0 (2024-05)

LTE;
5G;
Management and orchestration;
Management service template
(3GPP TS 32.160 version 18.5.0 Release 18)

Reference
RTS/TSGS-0532160vi50

Keywords
5G,LTE

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Intellectual Property Rights

Essential patents

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

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

Trademarks

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

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

Legal Notice

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

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

The cross reference between 3GPP and ETSI identities can be found under https://webapp.etsi.org/key/queryform.asp.

Modal verbs terminology

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

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

Contents

Foreword

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

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

Version x.y.z

where:

- x the first digit:
  - 1 presented to TSG for information;
  - 2 presented to TSG for approval;
  - 3 or greater indicates TSG approved document under change control.
- y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.
- z the third digit is incremented when editorial only changes have been incorporated in the document.

In the present document, certain 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" shall not to be used as 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 contains the templates to be used for the production of Management service component specifications type A, type B and type C [2].

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 28.533: "Management and orchestration; Architecture framework".

[3] 3GPP TS 32.156: "Telecommunication management; Fixed Mobile Convergence (FMC) Model Repertoire"

[4] ITU-T Recommendation M.3020 (07/2017): "Management interface specification methodology".

[5] 3GPP TR 21.801: "Specification drafting rules".

[6] 3GPP TS 28.622: "Telecommunication management; Generic Network Resource Model (NRM) Integration Reference Point (IRP); Information Service (IS)".

[7] 3GPP TS 28.541: "Management and orchestration; 5G Network Resource Model (NRM); Stage 2 and stage 3".

[8] 3GPP TS 32.302: "Telecommunication management; Configuration Management (CM); Notification Integration Reference Point (IRP); Information Service (IS)".

[9] 3GPP TS 32.300: "Telecommunication management; Configuration Management (CM); Name convention for Managed Objects".

[10] ITU-T Recommendation M.3020 (07/2011): "Management interface specification methodology" – Annex E "Information type definitions – type repertoire".

[11] IETF RFC 8407: "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models, October 2018".

[12] 3GPP TS 28.532: " Management and orchestration; Generic management services"

[13] IETF RFC 8528: "YANG Schema mount "

[14] OpenAPI: "OpenAPI 3.0.0 Specification", https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.1.md.

[15] draft-wright-json-schema-01 (October 2017): "JSON Schema: A Media Type for Describing JSON Documents".

[16] draft-wright-json-schema-validation-01 (October 2017: "JSON Schema Validation: A Vocabulary for Structural Validation of JSON".

[17] draft-wright-json-schema-hyperschema-01 (October 2017): "JSON Hyper-Schema: A Vocabulary for Hypermedia Annotation of JSON.

[18] IETF RFC 7950: "The YANG 1.1 Data Modeling Language, August 2016".

[19] IETF RFC 8525: " YANG Library".

[20] 3GPP TS 28.623: “Generic Network Resource Model (NRM) Integration Reference Point (IRP); Solution Set (SS) definitions”

3 Definitions of terms, symbols and abbreviations

3.1 Terms

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

3.2 Symbols

Void.

3.3 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].

C	Conditional
CM	Conditional Mandatory
CO	Conditional Optional
M	Mandatory
MnS	Management Service
NRM	Network Resource Model
O	Optional

4 Management service template (stage 1)

4.1 General

This template shall be used for the production of all requirement specifications for management and orchestration of 3GPP networks.

Instructions in *italics* below shall not be included in the requirements specifications.

Usage of fonts shall be according to the 3GPP drafting rules in TR 21.801 [5] for a TS (with some basic examples given in the 3GPP TS template).

4.2 Template for requirement specifications

X Management capabilities

X.a <Management capability name>

The management capability name above shall be replaced with the name of the management capability which is to be specified.

X.a.1 Description

For production of the contents of this clause, describe general information about the management capability.

X.a.2 Use cases

X.a.2.b <XXX Use case> <label>

For production of the contents of this clause, describe the motivation for one or more of the requirements in R4.c (referring to the requirement label(s)). The use case should also be labelled. The use case is not to clarify how to use a certain feature, and detailed sequence diagrams are not needed for a use case. The use case is to describe what are the benefits of the capability, what it is good for. High level diagrams including sequence diagrams may still be included if needed in order to better describe the use cases and motivate the corresponding requirements.

The format of the use case label is UC-xx-yy, where xx represents the abbreviation of the management capability name, yy is the serial number under the corresponding management capability category.

X.a.3 Requirements

For production of the contents of this subclause, describe the management capability requirements which are exposed to the consumer. Each requirement shall have a requirement label.

The format of the requirement label is REQ-xx-yy-zz, where xx is a unique abbreviation of the service/function, yy is MC (Management Capability) and zz is the serial number under the corresponding management capability category.

All requirements shall be motivated by either a use case or a textual motivation (also figures are allowed).

Requirement label	Description	Related use case(s)/Motivation
<REQ-xx-yy-zz>	<Requirement description>	<UC-xx-yy> / <Motivation text>

5 Management service template (stage 2)

5.1 General

5.1.1 General

The present document contains the templates to be used, for the production of all Management Service (MnS) specifications.

Clause 5.2 is applicable for specification of MnS component type B (NRM).

Clause 5.3 is applicable for specification of MnS component type A (operations and notifications) and type C (alarm and performance information).

The MnS template uses qualifiers M, O, CM, CO and C. The semantics of these qualifiers are defined in [3].

The MnS template uses type definition as one characteristic to describe class attributes and operation/notification parameters. The valid type definitions that can be used and their semantics are defined in [3].

Usage of fonts for the specific cases of class/attribute names etc., in addition to the general font requirements in the 3GPP drafting rules in 3GPP TR 21.801 [5], shall be according to the following table.

Table 5.1.1-1

Item	Font
Class names	Courier New
Attribute names	Courier New
Operation names	Courier New
Parameter names	Courier New
Assertion names	Courier New
Notification names	Courier New
Exception names	Courier New
State names	Arial
Matching Information	Courier New
Information Type	Courier New
Legal Values	Courier New
NOTE: These font requirements do not apply to UML diagrams.	

5.1.2 Management service components

A management service combines elements of management service components type A, B and C [1].

The template for NRM, see clause 5.2, applies to the specification of management service component type B.

The template for the Management service operations and notifications, see clause 5.3, applies to the specification of type A and type C.

5.2 Template for NRM

W4 Model

W4.1 Imported and associated information entities

W4.1.1 Imported information entities and local labels

This clause identifies a list of information entities (e.g. information object class, datatype, interface, attribute) that have been defined in other specifications and that are imported in the present (target) specification. All imported entities shall be treated as if they are defined locally in the target specification. One usage of import is for inheritance purpose.

Each element of this list is a pair (label reference, local label). The label reference contains the name of the original specification where the information entity is defined, the information entity type and its name. The local label contains the name of the information entity that appears in the target specification, and the entity name in the local label shall be kept identical to the name defined in the original specification. The local label may then be used throughout the target specification instead of that which appears in the label reference.

This information is provided in a table. An example of such a table is given here below:

Label reference	Local label
TS 28.622 [6], information object class, Top	Top
TS 28.541 [7] information object class NSI	NSI

W4.1.2 Associated information entities and local labels

This clause identifies a list of information entities (e.g. information object class, interface, attribute) that have been defined in other specifications and that are associated with the information entities defined in the present (target) specification. For the associated information entity, only its properties (e.g., DN (see TS 32.156 [3]), attribute (see TS 32.156 [3]) of an instance of the associated information entity) used as associated information needs to be supported locally in the target specification.

Each element of this list is a pair (label reference, local label). The label reference contains the name of the original specification where the information entity is defined, the information entity type and its name. The local label contains the name of the information entity that appears in the target specification. The local label may then be used throughout the target specification instead of that which appears in the label reference.

This information is provided in a table. An example of such a table is given here below:

Label reference	Local label
TS 28.541 [7], IOC, `GNBDUFunction`	`GNBDUFunction`

W4.2 Class diagram

W4.2.1 Relationships

This first set of diagrams represents all classes and datatypes defined in this MnS with all their relationships, including relationships with imported information entities (if any). These diagrams shall contain class cardinalities (for associations as well as containment relationships) and may also contain role names. These shall be UML compliant class diagrams (see also TS 32.156 [3]).

Characteristics (attributes, relationships) of imported information entities need not to be repeated in the diagrams. Allowable classes are specified in TS 32.156 [3].

*Use this as the first paragraph: "*This clause depicts the set of classes (e.g. IOCs) that encapsulates the information relevant for this MnS. This clause provides an overview of the relationships between relevant classes in UML. Subsequent clauses provide more detailed specification of various aspects of these classes."

W4.2.2 Inheritance

This second set of diagrams represents the inheritance hierarchy of all classes defined in this specification. These diagrams do not need to contain the complete inheritance hierarchy but shall at least contain the parent classes of all classes defined in the present document. By default, a class inherits from the class "top".

Characteristics (attributes, relationships) of imported classes need not to be repeated in the diagrams.

NOTE: some inheritance relationships presented in clause W4.2.2 may be repeated in clause W4.2.1 to enhance readability.

*Use "*This subclause depicts the inheritance relationships." *as the first paragraph.*

W4.3 Class definitions

Each class, with its stereotype name, is defined using the following structure.

Inherited items (attributes etc.) shall not be shown, as they are defined in the parent class(es) and thus valid for the subclass.

W4.3.a ClassName <<StereotypeName>>

StereotypeName is mandatory to be included in the clause header, except for the stereotype Information Object Class, for which it shall not be included in the clause header.

An example of a Class is `Subnetwork of stereotype Information Object Class.` *The heading of sub-clause W4.3.a for* `SubNetwork` *would look as follows:*

W4.3.a `SubNetwork`

An example of a Class is `SliceProfile of stereotype data type.` *The heading of W4.3.a for* `SliceProfile` *would look as follows:*

W4.3.a `SliceProfile <<dataType>>`

The various stereotypes can be found in TS 32.156 [3].

The "a" represents a number, starting at 1 and increasing by 1 with each new definition of a class.

W4.3.a.1 Definition

This clause is written in natural language. The <definition> clause refers to the class itself.

Classes (and datatypes) have a lifecycleStatus property as defined by [3] clause 5.2.A. If and only if the lifecycleStatus is not current (its default value), that shall be indicated in this clause.

Optionally, information on traceability back to one or more requirements supported by this class may be defined here, in the following form:

Referenced TS	Requirement label	Comment
TS 28.xyz [xy]	REQ-SM-CON-23	*Optional clarification*
TS 28.xyz [xy]	REQ-SM-FUN-11	*Optional clarification*

W4.3.a.2 Attributes

This clause presents the list of attributes, which are the manageable properties of the class. Each attribute is characterised by some of the attribute properties (see TS 32.156 [3]), i.e. supportQualifier (abbreviated by S), isReadable, isWritable, isInvariant and isNotifyable.

The legal values and their semantics for attribute properties are defined in TS 32.156 [3].

This information is provided in a table.

An example below indicates

Attribute name	S	isReadable	isWritable	isInvariant	isNotifyable
`eNodeBId`	M	T	F	T	T

Another example below indicates that the attribute `password1` *is not readable, is writable, is not an invariant and no* `notifyAttributeValueChange` *will be emitted when the attribute value is changed.*

Attribute name	S	isReadable	isWritable	isInvariant	isNotifyable
`password1`	O	F	T	F	F

Another example below indicates that the attribute `password2` *and* `password1` *(in example above) have the same qualifiers for the shown properties except that of isReadable. In the case of password1, the standard specification determines the qualifier to be M, i.e. it is readable. In the case of password2, the standard specification does not make a determination. The vendor would make the determination if the attribute is readable or not readable.*

Attribute name	S	isReadable	isWritable	isInvariant	isNotifyable
`password2`	O	O	T	F	F

In case there is one or more attributes related to role (see clause 5.2.9 of TS 32.156 [3]), the attributes related to role shall be specified at the bottom of the table with a divider "Attribute related to role", as shown in the following example:

Attribute name	S	isReadable	isWritable	isInvariant	isNotifyable
aTMChannelTerminationPointid	M	T	F	T	T
…					
…					
Attribute related to role					
theATMPathTerminationPoint	M	T	F	F	T
theIubLink	M	T	F	F	T

This clause shall state "None." when there is no attribute to define.

W4.3.a.3 Attribute constraints

This clause presents constraints for the attributes, and one use is to present the predicates for conditional qualifiers (CM/CO).

This information is provided in a table. An example of such a table is given here below:

Name	Definition
configuredMaxTxPower CM support qualifier	Condition: The sector-carrier has a downlink [4].
sNSSAIList CM support qualifier	Condition: Network slicing feature is supported [4]. LifecycleStatus of attribute: Deprecated.

Attributes have a lifecycleStatus property as defined by [3] clause 5.2.A. If and only if the lifecycleStatus is not current (its default value), that shall be indicated in this table.

This clause shall state "None." when there is no attribute constraint to define.

W4.3.a.4 Notifications

This clause, for this class, presents one of the following options:

a) The class *defines (and independent from those inherited) the support of a set of notifications that is identical to that defined in clause W4.5. In such case, use "The common notifications defined in clause W4.5 are valid for this class, without exceptions or additions." as the lone sentence of this clause.*

b) The class defines (and independent from those inherited) the support of a set of notifications that is a superset of that defined in clause W4.5. In such case, use "The common notifications defined in clause W4.5 are valid for this IOC. In addition, the following set of notification is also valid." as the lone paragraph of this clause. Then, define the 'additional' notifications in a table. See clause W4.5 for the notification table format.

c) The class defines (and independent from those inherited) the support of a set of notifications that is not identical to, nor a superset of, that defined in clause W4.5. In such case, use "The common notifications defined in clause W4.5 are not valid for this IOC. The set of notifications defined in the following table is valid." as the lone paragraph of this clause. Specify the set of notifications in a table. See clause W4.5 for the notification table format.

*d) The class does not define (and independent from those inherited) the support of any notification. In such case, use "There is n*o notification defined." as the lone sentence of this clause.

The notifications identified (i.e. option-a, option-b and option-c above) in this clause are notifications that may be emitted by the MnS producer, where the "object class" and "object instance" parameters of the notification header (see note 2) of these notifications identifies an instance of the class (or its direct or indirect derived class) defined by the encapsulating clause (i.e. clause W4.3.a).

The notifications identified (i.e. option-a and option-b above) in this clause, may originate from implementation object(s) whose identifier may or may not be the same as that carried in the notification parameters "object class" and "object instance". Hence the identification of notifications in this clause does not imply nor identify those notifications as being originated from an instance of the class (or its direct or indirect derived class) defined by the encapsulating clause (i.e. clause W4.3.a).

This clause shall state "This class does not support any notification." (see option-c) when there is no notification defined for this class. (Note that if its parent class has defined some notifications, the implementation of this class is capable of emitting those inherited defined notifications.)

The notification header is defined in TS 32.302 [8].

The qualifier of a notification, specified in Notification Table, indicates if an implementation may generate a notification carrying the DN of the subject class.

An MnS consumer may receive notification-XYZ that carries DN (the "object class" and "object instance") of class-ABC instance if and only if:

a) The class-ABC Notification Table defines the notification-XYZ and

b) The class-ABC instance implementation supports this notification-XYZ and

c) An MnS defines the notification-XYZ and

d) The MnS implementation supports this notification-XYZ.

W4.3.a.5 State diagram

This subclause contains state diagrams. A state diagram of an information object class defines permitted states of this information object class and the transitions between those states. A state is expressed in terms of individual attribute values or a combination of attribute values or involvement in relationships of the information object class being defined. This shall be a UML compliant state diagram.

This subclause shall state "None." when there is no State diagram defined.

W4.5 Attribute definitions

W4.5.1 Attribute properties

It has a lone paragraph "The following table defines the properties of attributes that are specified in the present document. ".

Each information attribute is defined using the following structure.

Inherited attributes shall not be shown, as they are defined in the parent class(es) and thus valid for this class.

An attribute has properties (see TS 32.156 [3]). Some properties of an attribute are defined in W4.3.a.2 (e.g. Support Qualifier). The remaining properties of an attribute (e.g. documentation, default value) are defined here.

The information is provided in a table. In case a) attributes of the same name are specified in more than one class and b) the attributes have different properties, then the attribute names (first column) should be prefixed with the class name followed by a period.

An example is given below:

Attribute Name	Documentation and Allowed Values	Properties
xyzId	It identifies … allowedValues: …	type: Integer multiplicity: … isOrdered: … isUnique: … defaultValue: … isNullable: False
Abc.state	It indicates … allowedValues: "ON": the state is on; "OFF": the state is off.	type: <<enumeration>> multiplicity: 1 isOrdered: N/A isUnique: N/A defaultValue: False isNullable: False
Zyz.state	It indicates … allowedValues: "HIGH": the state is high; "MEDIUM": the state is medium; "LOW": the state is low.	type: <<enumeration>> multiplicity: 1 isOrdered: N/A isUnique: N/A defaultValue: False isNullable: False
abc	It defines… allowedValues: …	type: … multiplicity: … isOrdered: … isUnique: … defaultValue: … isNullable: …

In case there is one or more attributes related to role (see clause 5.2.9 of TS 32.156 [3]), the attributes related to role shall be specified at the bottom of the table with a divider "Attribute related to role". See example below.

Attribute Name	Documentation and Allowed Values	Properties
abc	It defines… allowedValues: …	type: PlmnId multiplicity: … isOrdered: … isUnique: … defaultValue: … isNullable: …
Attribute related to role		
aEnd	It defines… allowedValues: Values to be conformant to TS 32.300 [9] …	type: DN multiplicity: … isOrdered: … isUnique: … defaultValue: … isNullable: False

This clause shall state "None." if there is no attribute to define.

W4.5.2 Constraints

This clause indicates whether there are any constraints affecting attributes. Each constraint is defined by a triplet (propertyName, affectedAttributes, propertyDefinition). PropertyDefinitions are expressed in natural language.

An example is given here below:

Name	Affected attribute(s)	Definition
inv_TimerConstraints	ntfTimeTickTimer	The ntfTimeTickTimer is lower than or equal to ntfTimeTick.

This clause shall state "None." if there is no constraint.

W4.6 Common notifications

This clause presents notifications that may be referred to by any class defined in the specification. This information is provided in tables.

W4.6.1 Alarm notifications

The following quoted text shall be copied as the only paragraph of this clause.

"This clause presents a list of notifications, defined in TS 28.532 [12], that an MnS consumer may receive. The notification header attribute `objectClass/objectInstance`, defined in TS 28.541 [7], shall capture the DN of an instance of a class defined in the present document."

The information is provided in a table. The following is an example.

Name	S	Notes
`notifyNewAlarm`	M	--

W4.6.2 Configuration notifications

The following quoted text shall be copied as the only paragraph of this clause.

"This clause presents a list of notifications, defined in TS 28.532 [12], that an MnS consumer may receive. The notification header attribute `objectClass/objectInstance`, defined in TS 32.302 [8], shall capture the DN of an instance of a class defined in the present document."

The information is provided in a table. The following is an example.

Name	S	Notes
`notifyMOIAttributeValueChange`	O	--
`notifyMOICreation`	O	--
`notifyMOIDeletion`	O	--

W4.6.3 Threshold Crossing notifications

The following quoted text shall be copied as the only paragraph of this clause.

*"*This clause presents a list of notifications, defined in TS 28.532 [12], that an MnS consumer may receive. The notification header attribute `objectClass/objectInstance`, defined in TS 28.541 [7], shall capture the DN of an instance of a class defined in the present document."

The information is provided in a table. The following is an example.

Name	S	Notes
`notifyThresholdCrossing`	O	

5.3 Template for Management service operations and notifications

---

Y4 Overview

Yb Management service name

Management service name should be replaced with the name of the Management Service (MnS).

"b" represents a number, starting at 1 and increasing by 1 with each new definition of a Management Service.

Yb.1 Operations and notifications

Yb.1.a Operation OperationName

OperationName is the name of the operation followed by a qualifier indicating whether the operation is Mandatory (M), Optional (O), Conditional-Mandatory (CM), Conditional-Optional (CO), or SS-Conditional (C).

"a" represents a number, starting at 1 and increasing by 1 with each new definition of an operation.

Yb.1.a.1 Definition

Yb.1.a.1.1 Description

This subclause shall be written in natural language.

Operations have a lifecycleStatus property as defined by [3] clause 5.2.A. If and only if the lifecycleStatus is not current (its default value), that shall be indicated in this subclause.

Information on traceability back to one or more requirements supported by this operation should also be defined here, in the following form:

Referenced TS	Requirement label	Comment
3GPP TS 32.xyz [xy]	REQ-SM-CON-23	*Optional clarification*
3GPP TS 32.xyz [xy]	REQ-SM-FUN-11	*Optional clarification*

Yb.1.a.1.2 Pre-condition

A pre-condition is a collection of assertions joined by AND, OR, and NOT logical operators. The pre-condition shall be true before the operation is invoked. An example is given here below:

`notificationCategoriesNotAllSubscribed` OR
`notificationCategoriesParameterAbsentAndNotAllSubscribed`

Each assertion is defined by a pair (propertyName, propertyDefinition). All assertions constituting the pre-condition are provided in a table. An example of such a table is given here below:

Assertion Name	Definition
`notificationCategoriesNotAllSubscribed`	At least one `notificationCategory` identified in the `notificationCategories` input parameter is supported by an `MnS producer` and is not a member of the `ntfNotificationCategorySet` attribute of an `NtfSubscription` which is involved in a subscription relationship with the `NtfSubscriber` identified by the `managerReference` input parameter.
`notificationCategoriesParameterAbsentAndNotAllSubscribed`	The `notificationCategories` input parameter is absent and at least one `notificationCategory` supported by `MnS producer` is not a member of the `ntfNotificationCategorySet` attribute of an `ntfSsubscription` which is involved in a subscription relationship with the `NtfSubscriber` identified by the `managerReference` input parameter.

Yb.1.a.1.3 Post-condition

A post-condition is a collection of assertions joined by AND, OR, and NOT logical operators. The post-condition shall be true after the completion of the operation. When nothing is said in a post-condition regarding an information entity, the assumption is that this information entity has not changed compared to what is stated in the pre-condition. An example is given here below:

`subscriptionDeleted` OR `allSubscriptionDeleted`

Each assertion is defined by a pair (propertyName, propertyDefinition). All assertions constituting the post-condition shall be provided in a table. An example of such a table is given here below:

Assertion Name	Definition
`subscriptionDeleted`	The `ntfSubscription` identified by `subscriptionId` input parameter is no more involved in a subscription relationship with the `ntfSubscriber` identified by the `managerReference` input parameter and has been deleted. If this `ntfSubscriber` has no more `ntfSubscription`, it is deleted as well.
`allSubscriptionDeleted`	In the case `subscriptionId` input parameter was absent, the `ntfSubscriber` identified by the `managerReference` input parameter is no more involved in any subscription relationship and is deleted, the corresponding `ntfSubscription` have been deleted as well.

Yb.1.a.1.4 Exceptions

List of exceptions that can be raised by the operation. Each element shall be a tuple (exceptionName, condition, ReturnedInformation, exitState).

Yb.1.a.1.4.c exceptionName

ExceptionName is the name of an exception.

"c" represents a number, starting at 1 and increasing by 1 with each new definition of an exception.

This information shall be provided in a table. An example of such a table is given here below:

Exception Name	Definition
`ope_failed_existing_subscription`	**Condition:** (`notificationCategoriesNotAllSubscribed` OR `notificationCategoriesParameterAbsentAndNotAllSubscribed`) not verified. **Returned information:** output parameter status is set to `OperationFailedExistingSubscription`. **Exit state:** Entry State.

NOTE: An example of an exception can be a situation where an operation is raised and the required information between a consumer and producer cannot be conveyed via the input and output parameters.

Yb.1.a.2 Input parameters

List of input parameters of the operation. Each element shall be a tuple (Parameter Name, Support Qualifier, Information Type (see [10] and note 1) and an optional list of Legal Values supported by the parameter, Comment). Legal Values for the Support Qualifier are: Mandatory (M), Optional (O), Conditional-Mandatory (CM), Conditional-Optional (CO), or SS-Conditional (C).

This information shall be provided in a table. An example of such a table is given here below:

Parameter Name	S	Information Type / Legal Values	Comment
`eventIdList`	M	`SET OF INTEGER / --`	One or more event identifiers

NOTE: Information Type qualifies the parameter of Parameter Name. In the case where the Legal Values can be enumerated, each element is a pair (Legal Value Name, Legal Value Semantics), unless a Legal Value Semantics applies to several values in which case the definition can be provided only once. When the Legal Values cannot be enumerated, the list of Legal Values is defined by a single definition.

Yb.1.a.3 Output parameters

List of output parameters of the operation. Each element tuple (Parameter Name, Support Qualifier, Matching Information / Information Type (see [10]) (Note 1) and an optional list of Legal Values supported by the parameter, Comment). Legal Values for the Support Qualifier are: Mandatory (M), Optional (O), Conditional-Mandatory (CM), Conditional-Optional (CO), or SS-Conditional (C).

This information shall be provided in a table. An example of such a table is given here below:

Parameter Name	S	Matching Information / Information Type / Legal Values	Comment
`eventTime`	M	`AlarmInformation.alarmRaisedTime / GeneralizedTime / --`	The parameter carries the • alarmRaisedTime in case notificationType carries notifyNewAlarm, • alarmChangedTime in case notificationType carries notifyChangedAlarm, • alarmClearedTime in case notificationType carries notifyClearedAlarm.

NOTE: Information Type qualifies the parameter of Parameter Name. In the case where the Legal Values can be enumerated, each element is a pair (Legal Value Name, Legal Value Semantics), unless a Legal Value Semantics applies to several values in which case the definition can be provided only once. When the Legal Values cannot be enumerated, the list of Legal Values is defined by a single definition.

This table shall also include a special parameter 'status' to indicate the completion status of the operation (success, partial success, failure reason etc.).

Yb.1.a.4 Result

Yb.1.a.4,1 Error messages

This subclause presents error messages in case the operation is not successful.

This subclause does not need to be present when there are no error messages to define.

Yb.1.a.4,2 Constraints

This subclause presents constraints for the operation or its parameters.

This subclause does not need to be present when there are no constraints to define.

Yb.1.a Notification NotificationName

NotificationName shall be the name of the notification followed by a qualifier indicating whether the notification is Mandatory (M), Optional (O), Conditional-Mandatory (CM), Conditional-Optional (CO) or SS-Conditional (C).

"a" represents a number, starting at 1 and increasing by 1 with each new definition of a notification.

Yb.1.a.1 Definition

This subclause shall be written in natural language.

Notifications have a lifecycleStatus property as defined by [3] clause 5.2.A. If and only if the lifecycleStatus is not current (its default value), that shall be indicated in this subclause.

Information on traceability back to one or more requirements supported by this notification should also be defined here, in the following form:

Referenced TS	Requirement label	Comment
3GPP TS 32.xyz [xy]	REQ-SM-CON-23	*Optional clarification*
3GPP TS 32.xyz [xy]	REQ-SM-FUN-11	*Optional clarification*

Yb.1.a.2 Input parameters

List of input parameters of the notification. Each element is a tuple (Parameter Name, Qualifiers, Matching Information / Information Type (see [10]) (Note 1) and an optional list of Legal Values supported by the parameter, Comment).

The column "Qualifiers" contains the two qualifiers, Support Qualifier and Filtering Qualifier, separated by a comma. The Support Qualifier indicates whether the attribute is Mandatory (M), Optional (O), Conditional-Mandatory (CM), Conditional-Optional (CO), or SS-Conditional (C).

This information shall be provided in a table. An example of such a table is given here below:

Parameter Name	S	Matching Information / Information Type / Legal Values	Comment
managerReference	M	ntfSubscriber.ntfManagerReference / STRING / --	It specifies the reference of the consumer to which notifications shall be sent.
alarmType	M	AlarmInformation.eventType / ENUMERATED / "Communications Alarm": a communication error alarm. "Processing Error Alarm": a processing error alarm. "Environmental Alarm": an environmental violation alarm. "Quality Of Service Alarm": a quality of service violation alarm. "Equipment Alarm": an alarm related to equipment malfunction.	

NOTE: Information Type qualifies the parameter of Parameter Name. In the case where the Legal Values can be enumerated, each element is a pair (Legal Value Name, Legal Value Semantics), unless a Legal Value Semantics applies to several values in which case the definition can be provided only once. When the Legal Values cannot be enumerated, the list of Legal Values is defined by a single definition.

Yb.1.a.3 Triggering event

The triggering event for the notification to be sent is the transition from the information state defined by the "from state" subclause to the information state defined by the "to state" subclause.

Yb.1.a.3.1 From state

This subclause is a collection of assertions joined by AND, OR, and NOT logical operators. An example is given here below:

alarmMatched AND alarmInformationNotCleared

Each assertion is defined by a pair (propertyName, propertyDefinition). All assertions constituting the state "from state" are provided in a table. An example of such a table is given here below:

Assertion Name	Definition
alarmMatched	The matching-criteria-attributes of the newly generated network alarm has values that are identical (matches) with ones in one AlarmInformation in AlarmList.
alarmInformationNotCleared	The perceivedSeverity of the newly generated network alarm is not Cleared.

Yb.1.a.3.2 To state

This subclause contains a collection of assertions joined by AND, OR and NOT logical operators. When nothing is said in a to-state regarding an information entity, the assumption is that this information entity has not changed compared to what is stated in the from-state. An example is given here below:

resetAcknowledgementInformation AND perceivedSeverityUpdated

Each assertion is defined by a pair (propertyName, propertyDefinition). All assertions constituting the state "to state" are provided in a table. An example of such a table is given here below:

Assertion Name	Definition
resetAcknowledgementInformation	The matched AlarmInformation identified in inv_alarmMatched in pre-condition has been updated according to the following rule: ackTime, ackUserId and ackSystemId are updated to contain no information; ackState is updated to "unacknowledged".
perceivedSeverityUpdated	The perceivedSeverity attribute of matched AlarmInformation identified in inv_alarmMatched in pre-condition has been updated.

Yb.2 Managed information

6 NRM Stage 3 definition rules

6.1 Mappings from stage 2 artefacts to stage 3 JSON schema

6.1.1 Usage of JSON schema

JSON schema is used to describe a set of valid schema documents sent over the wire in HTTP request and response messages of the ProvMnS. JSON schema does not describe the concrete implementation of the NRM on the producer.

Definitions are written in YAML.

6.1.2 Concrete NRM classes

A NRM class (managed object class) is represented by a JSON object. The properties of the JSON object are the NRM class attributes and the name contained NRM classes.

YAML schema	YAML document example
type: object properties: {}	{}

In the following example the class contains an "attributeA" of type "string" and an "attributeB" of type "number".

YAML schema	YAML document example

type: object
properties:
  attributeA:
  type: string
  attributeB:
  type: number
	attributeA: ABC
	attributeB: 45

The JSON object representing the class instance is preceded by a key equal to the class name.

In the following example the class name is "classA". Attributes are omitted for the sake of simplicity.

YAML schema	YAML document example

type: object
properties:
  classA:
  type: object
  properties: {}
	classA: {}

Multiple managed object instances of the same class are represented using a JSON array, where each item of the array is a JSON object with a managed object class instance representation.

YAML schema	YAML document example

type: object
properties:
  ClassA:
	ClassA:
	  - {}
	  - {}

`    type: array`
`    items:`
`      type: object`
`      properties: {}`
	`  - {}`

6.1.3 Abstract classes

Abstract classes shall be defined in a "definitions" object and referenced in the schema of the concrete class using the "$ref" keyword.

In the following example the abstract class can be instantiated zero or one time..

YAML schema	YAML document example

`definitions:`
`  ClassA-Abstract:`
`    type: object`
`    properties: {}`
`type: object`
`properties:`
`  ClassA:`
`    $ref: '#/definitions/ClassA-Abstract'`
	`ClassA: {}`

In the following example the abstract class can be instantiated zero or more times.

YAML schema	YAML document example

`definitions:`
`  ClassA-Abstract:`
`    type: object`
`    properties: {}`
`type: object`
`properties:`
`  ClassA:`
`    type: array`
`    items:`
`      $ref: '#/definitions/ClassA-Abstract'`
	`ClassA:`
	`  - {}`
	`  - {}`
	`  - {}`

Abstract classes can be defined as well in separate files. Assume a file with the name "myDefs.json" includes the "definitions" object with the definition of "ClassA-Abstract ".

YAML schema	YAML document example

`definitions:`
`  ClassA-Abstract:`
`    type: object`
`    properties: {}`	

The definition of "ClassA-Abstract" is then referenced like

YAML schema	YAML document example

`type: object`
`properties:`
`  ClassA:`
`    type: array`
`    items:`
`      $ref: 'myDefs.json#/definitions/ClassA-Abstract'`
	`ClassA:`
	`  - {}`
	`  - {}`
	`  - {}`

6.1.4 Name containment

Name contained NRM class instances are modeled as property of the containing class. The name of the property is the class name. The value is an array with manged object class representations of that class. Cardinality of the name containment relationship is specified using the "minItems" and "maxItems" keywords.

If the maximum number of items is unbounded, the "maxItems" keyword shall be omitted. If the minimum number of items is 0, the "minItems" keyword can be omitted.

The contained class shall not be listed as required property. This allows omitting the property representing the contained class instances completely in a JSON document instead of having an empty array.

In the following example an instance of "classA" name contains 1…1000 instances of "classB".

YAML schema	YAML document example

type: object
properties:
  ClassA:
  type: array
  items:
  type: object
  properties:
  ClassB:
  type: array
  minItems: 1
  maxItems: 1000
  items:
  type: object
  properties: {}
	ClassA:
	  - ClassB:
	  - {}
	  - {}

Managed objects class instances of more than one class can be name contained.

YAML schema	YAML document example

type: object
properties:
  ClassA:
  type: array
  items:
  type: object
  properties:
  ClassB:
  type: array
  items:
  type: object
  properties: {}
  ClassC:
  type: array
  items:
  type: object
  properties: {}
	ClassA:
	  - ClassB:
	  - {}
	  - {}
	  - ClassC:
	  - {}
	  - {}
	  - {}

The contained managed object classes may be defined as abstract classes first, and then referenced.

YAML schema	YAML document example

definitions:
  ClassB-SingleAbstract:
  type: object
  properties: {}
  ClassC-SingleAbstract:
  type: object
  properties: {}
type: object
properties:
  ClassA:
  type: array
  items:
  type: object
  properties:
  ClassB:
  type: array
  items:
  $ref: '#/definitions/ClassB-SingleAbstract'
  ClassC:
  type: array
  items:
  $ref: '#/definitions/ClassC-SingleAbstract'
	ClassA:
	  - ClassB:
	  - {}
	  - {}
	  - ClassC:
	  - {}
	  - {}
	  - {}

or, when the abstract class is defined as an array, then

YAML schema	YAML document example

```
definitions:
  ClassB-MultipleAbstract:
    type: array
    items:
      type: object
      properties: {}
  ClassC-MultipleAbstract:
    type: array
    items:
      type: object
      properties: {}
type: object
properties:
  ClassA:
    type: array
    items:
      type: object
      properties:
        ClassB:
          $ref: '#/definitions/ClassB-MultipleAbstract'
        ClassC:
          $ref: '#/definitions/ClassC-MultipleAbstract'
```

```
ClassA:
  - ClassB:
      - {}
      - {}
  - ClassC:
      - {}
      - {}
      - {}
```

6.1.5 Recursive name containment

Classes may name contain themselves. This shall be modeled in JSON schema with recursion. Recursion requires using a "definitions" object with the definition of an abstract class.

In the following example each instance of "classA" contains zero or one instance of "classA".

YAML schema

```
definitions:
  ClassA-Abstract:
    type: object
    properties:
      classA:
        $ref: '#/definitions/ClassA-Abstract'
type: object
properties:
  ClassA:
    $ref: '#/definitions/ClassA-Abstract'
```

YAML document example

```
ClassA:
  ClassA:
    ClassA: {}
```

In the following example each instance of "classA" contains zero or more instances of "classA".

YAML schema

```
definitions:
  ClassA-MultipleAbstract:
    type: array
    items:
      type: object
      properties:
        classA:
          $ref: '#/definitions/ClassA-MultipleAbstract'
type: object
properties:
  ClassA:
    $ref: '#/definitions/ClassA-MultipleAbstract'
```

YAML document example

```
ClassA:
  - ClassA:
      - {}
      - {}
  - ClassA:
      - ClassA:
          - {}
```

6.1.6 Inheritance

JSON schema does not have the concept of inheritance. Inheritance can be emulated by the composition of schemas with the "allOf" keyword.

In the following example the attribute "attrB" is added to the attribute "attrA" of "classA-Abstract" to construct "ClassB".

YAML schema

```
definitions:
```

YAML document example

```
ClassB:
```

```
  ClassA-Abstract:
    type: object
    properties:
      attrA:
        type: string
type: object
properties:
  ClassB:
    type: array
    items:
      allOf:
        - $ref: '#/definitions/ClassA-Abstract'
        - type: object
          properties:
            attrB:
              type: number
```

```
  - attrA: ABC
    attrB: 5
  - attrA: DEF
    attrB: 4
  - attrA: GHI
    attrB: 23
```

The other possibility is to specify the inherited attribute directly along with the added attributes, thus having no inheritenace or any emulation thereof in NRM stage 3 definitions.

6.1.7 NRM class naming attribute "id"

The naming attribute "id" is mapped to a required property of the class object, where the key is "id" and the type is "string".

YAML schema

```
type: object
properties:
  ClassA:
    type: array
    items:
      type: object
      properties:
        id:
          type: string
        required:
          - id
```

YAML document example

```
ClassA:
  - id: '1'
  - id: '2'
  - id: '3'
```

6.1.8 NRM class attributes

NRM class attributes other than the naming attribute "id" shall be carried as properties in an "attributes" object.

YAML schema

```
type: object
properties:
  classA:
    type: array
    items:
      type: object
      properties:
        id:
          type: string
        attributes:
          type: object
          properties: {}
        required:
          - id
```

YAML document example

```
classA:
  - id: '1'
    attributes: {}
  - id: '2'
    attributes: {}
  - id: '3'
    attributes: {}
```

The class attributes are name/value pairs (properties) of the "attributes" object.

6.1.9 Vendor specific extensions

Vendor-specific attributes shall be added to standardized JSON schemas using the mechanism in clause 6.1.6 "Inheritance".

6.1.10 Attribute support qualifier

The attribute support qualifier is defined in clause 6 of TS 32.156 [3]. This qualifier specifies a requirement for the MnS producer.

Attributes may or may not be present in a JSON document carried in a HTTP request or response message, no matter what their support qualifier in the NRM is. For this reason, no qualification is required for attributes in the JSON schema for NRMs. By default, the properties defined by the "properties" keyword are not required and can be omitted in a document instance.

However, some attributes like the "id" naming attribute shall be always present when a managed object class instance is carried in a HTTP request or response. These attributes shall be listed as array items in the value of the "required" keyword.

YAML schema	YAML document example

type: object
properties:
  classA:
  type: array
  items:
  type: object
  properties:
  id:
  type: string
  required:
  - id
	classA:
	  - id: '1'
	  - id: '2'
	  - id: '3'

6.1.11 Attribute properties

6.1.11.1 Introduction

The attribute properties are defined in clause 5.2.1.1 of TS 32.156 [3]. They reflect properties of the attributes exhibited by the MnS producer. Their purpose is not to specify requirements for the attribute when transferred over the wire. For this reason, care should be taken when mapping attribute properties to JSON schema keywords.

6.1.11.2 Attribute property "multiplicity"

Attributes of scalar type with multiplicity equal to "1" are mapped to a name/value pair whose value is either a number, a string or one of the literal names false, null or true.

Attributes of scalar type with nultiplicity bigger than "1" are mapped to a name/value pair whose value is a JSON array, and the array items are either a number, a string or one of the literal names false, null or true.

Attributes of structured type with multiplicity equal to "1" are mapped to a single name/value pair whose value is a JSON object, whose properties are described by the structured data type.

Attributes of structured type with multiplicity greater than "1" are mapped to a name/value pair whose value is a JSON array, and the items are JSON objects, whose properties are described by the structured data type.

6.1.11.3 Attribute property "isUnique"

The semantics of his attribute property is mapped to the "uniqueItems" keyword with a value set to true.

```
properties:
  flower:
    type: array
    uniqueItems: true
    items:
      type: string
```

6.1.11.4 Attribute property "isOrdered"

This attribute property is a requirement for the MnS producer and not mapped to any JSON schema keyword.

6.1.11.5 Attribute property "defaultValue"

This attribute property is a requirement for the MnS producer and not mapped to any JSON schema keyword.

NOTE: The OpenApi Specification [14] defines the "default" keyword. This default value represents what would be assumed by the consumer of the input as the value of the schema if a value is not provided in the consumed JSON instance document. The sematics of default in the OpenApi Specification [14] is hence different from the semantics of default in TS 32.156 [3].

6.1.11.6 Attribute property "isNullable"

The semantics of this attribute property is mapped to the "nullable" keyword with a value set to true.

Example:

```
properties:
  flower:
    type: string
    nullable: true
```

NOTE: The "nullable" keyword is defined only in the OpenApi Specification [14]. JSON schema as defined in [15], [16], [17] does not specify this keyword.

6.1.11.7 Attribute property "isInvariant"

This attribute property is a requirement for the MnS producer and not mapped to any JSON schema keyword.

6.1.11.8 Attribute property "isReadable" and "isWritable"

The semantics of these properties are mapped to the "readOnly" and "writeOnly" keywords with the values set according to the following table. The default value of the "readOnly" and "writeOnly" keywords is boolean "false".

Stage 2 statement	Stage 2 semantic	Stage 3 statements	Stage 3 semantic
isReadable=True (default) isWritable=True (default)	Attribute can be read. Attribute can be written.	readOnly=False (default) writeOnly=False (default)	Attribute can be read. Attribute can be written.
isReadable=True (default) isWritable=False	Attribute can be read. Attribute cannot be written.	readOnly=True writeOnly=False (default)	Attribute can be read. Attribute cannot be written.
isReadable=False isWritable=True (default)	Attribute cannot be read. Attribute can be written.	readOnly=False (default) writeOnly=True	Attribute cannot be read. Attribute can be written.
isReadable=False isWritable=False	Attribute cannot be read. Attribute cannot be written.	readOnly=True writeOnly=True	Attribute cannot be read. Attribute cannot be written.

If "writeOnly" for an attribute has a value of boolean "true", it indicates that the attribute shall never be present in instance documents sent by the MnS producer to the MnS consumer.

If "readOnly" for an attribute has a value of boolean "true", it indicates that the attribute shall never be present in instance documents sent by the the MnS consumer to the MnS producer.

Example:

```
properties:
  flower:
    type: string
    readOnly: true
    writeOnly: false
```

6.1.11.9 Attribute property "isNotifyable"

This attribute property is a requirement for the MnS producer and not mapped to any JSON schema keyword.

6.1.11.10 Attribute property "allowedValues"

Allowed values for "string" are specified using the "minLength", "maxLength" and "pattern" keywords.

Allowed values for "number" and "integer" are specified using the "multipleOf", "maximum", "exclusiveMaximum", "minimum" and "exclusiveMinimum" keywords.

Allowed values of any type can be restricted by using the "enum" and "const" keywords.

6.1.11.11 Attribute property "lifecycleStatus"

LifecycleStatus=current is the default case so it is not mapped to any JSON schema keyword.

LifecycleStatus=deprecated shall be mapped the "deprecated" keyword with a value of true.

6.2 Stage 3 YANG style and example

The next clause defines general rules for YANG modules. The following clauses specify how specific Stage to constructs should be mapped to YANG. Each clause may include the following clauses:

- The clause of Reference [3] for which mapping is specified.
- An example model that will be mapped.
- Mapping rules.
- An example of the resulting YANG statements.

6.2.1 General Modeling Rules

6.2.1.1 Modeling Resources

Resources shall be modeled as YANG data nodes (leaf, leaf-list, container, list) instead of Classes and Attributes. Specific operations shall be modelled as YANG actions.

6.2.1.2 Unique YANG Module names

The names of 3GPP YANG modules shall start with the "`_3gpp`" prefix.

6.2.1.3 Unique YANG Namespace

The namespace of a 3GPP YANG module's namespace shall have the following form:

```
urn:3gpp:saX:<module-name>
```

saX denotes the group creating the relevant YANG model e.g. "sa5"

Reference: https://tools.ietf.org/html/rfc8407#section-4.9 [11].

6.2.1.4 Unique YANG Module Prefixes

3GPP YANG Modules shall use prefixes ending with "3gpp". Prefixes should be short preferably not longer than 10 characters but 13 characters at most.

e.g. prefix nrmtype -> prefix nrmtype3gpp

NOTE: To ensure that the prefix (in the yang prefix statement) is globally unique a prefix-suffix is used. While global uniqueness of prefixes is not mandatory most SW implementations have problems and need workarounds in case conflicting prefixes are found.

6.2.1.5 Use YANG version 1.1

YANG version 1.1 shall be used. See [18].

6.2.1.6 YANG constructs not to be used – not recommended

The following YANG constructs shall not be used in 3GPP YANG models as they are not available in the Stage 2 modeling terminology, thus not needed.

- anyxml
- rpc – use actions instead
- deviation

The following YANG statements should not be used in 3GPP YANG models:

- anydata. Whenever possible data should be modeled with list, leaf-list, leaf data nodes. In the rare case where the type of an attribute is unknown (E.g., a an attribute that can be of any attribute type) the YANG “anyadata” statement may be used.

6.2.1.7 Reuse standards from other standard organizations

Whenever there is a suitable existing standard from another standard organization or industry forum its usage should be preferred before defining a 3GPP model covering the same scope. E.g. ietf-types, ietf-inet-types

3GPP models shall link to and reference YANG models from other standard organizations/industry forum whenever applicable.

6.2.1.8 Vendor specific model changes

Vendors shall not modify 3GPP YANG modules either by changing the original file or by adding vendor specific YANG modules that contain deviations targeting parts of a 3GPP module. Only the following exceptions are allowed from the above rule:

- Deviations that maintain backwards compatibility as defined in RFC 7950 [18] are allowed
- Marking as "not supported" any model element that is optional to support as defined by the 3GPP stage 2 supportQualifier is allowed.

Vendors extensions to the model shall be done in separate YANG modules; they do not impact compliance.

Vendor extensions to the model should follow the IOC/attribute structure based on TS 32.156[3] and the mapping defined in clause 6.2 and its subclauses. Inheritance from abstract 3GPP IOCs (e.g. Top) is encouraged.

Example 1 – Add a vendor specific attribute to a 3GPP specified IOC:

```
augment /me3gpp:ManagedElement/attributes {
  leaf isCabinetClosed {
    type boolean ;
    description “Indicates whether the doors of the HW cabinet is closed.” ;
  }
}
```

Example 2 – Add a vendor specific IOC:

```
//vendor class
grouping VendorClassGrp {
  // contains all attributes
  leaf exampleAttribute {
    type string;
    description vendorMarker;
  }
}

augment /me3gpp:ManagedElement {
```

```
  list VendorClass {
    key id;
    uses top3gpp:Top_Grp;
    container attributes {
        uses VendorClassGrp ;
     }
    //YANG lists representing contained classes
  }
}
```

6.2.1.9 Model correctness, checking

3GPP YANG modules shall be checked with the pyang tool. See: PYANG an extensible YANG validator and converter [x].

The "pyang –-strict" command shall be run with no errors returned.

"pyang —lint" should also be run against all 3GPP YANG modules. Errors and warning produced by the "pyang –lint" checks should be removed. However, as these errors/warnings do not affect the corretness or functionality of the YANG module, and in some cases the changes needed to remove them would actually degrade readability, it is not a required to remove the errors/warnings produced by the "pyang –lint".

6.2.1.10 YANG modules in technical specifications

If a module's text is included in a technical specification, each YANG module shall be contained in a separate clause. The clause's title shall not include the revision date of the module.

To facilitate automatic code extraction from the MS Word specification:

- Immediately before the first line of a YANG module/submodule a line should be inserted containing only the text

<CODE BEGINS>

- the module's first statement shall start with the keyword "module" (or submodule) in the first place (no whitespace allowed before it on the line).
- followed by a single space.
- followed by the name of the module/submodule.
- followed by a single space and an opening curly bracket "{".
- All following lines shall be indented at least with two spaces.
- the last line of the module shall be a single "}" without any characters before or after it (especially no white space before it)
- Immediately after the last line of a YANG module/submodule a line should be inserted containing only the text

<CODE ENDS>

6.2.1.11 Module header statements

A module's organization and description statements shall be present. The organization shall include the string "3GPP".

A module shall contain the following contact statement:

```
  contact "https://www.3gpp.org/DynaReport/TSG-WG--S5--officials.htm?Itemid=464;"
```

6.2.1.12 Provide description and reference statements

A "description" statement should be present for each YANG schema node. As an exception: for individual leafs, leaf-lists, enums, case statements, typedef statements, where the schema node's name describes the node sufficiently, the "description" may be omitted.

A "reference"substatement to the module statement shall be present that specifies the technical specification where the YANG module is defined. In order to easily list with a "grep" command YANG modules belonging to a specific technical specification, the format of the first line of this reference statement shall start exactly with:

- new-line followed by
- the string ‘ reference ”3GPP TS ‘

  (that is 2 leading spaces + reference + 1 space + a double quote + 3GPP TS + 1 more space) followed by
- the number of the technical specification.

E.g." reference "3GPP TS 28.622".

6.2.1.13 YANG module revisions

A YANG module version is identified by its name and the date in the latest revision statement in it. When a module is changed in any way a new revision statement/date shall be added to it. Different versions of the same module shall contain a different (latest) revision date.

In order to minimize changes to the YANG interface it is recommended to use the same module revision (same YANG file) in multiple 3GPP releases as long as there is no interface effecting change between the different releases for that YANG module.

A separate "revision" statement shall be present for each new published version of a module. The revision statement shall contain a reference substatement listing the numbers of all 3GPP change requests and any other documents that resulted in the creation of the new revision.

Example:

```
revision 1956-10-13 {
  reference “CR-0258, CR-0267”;}
```

NOTE: Void.

If multiple change requests modify the new revision of a YANG module, the content of the reference substatements should be merged.

In case a YANG module revision (same YANG file) is used in multiple releases and needs similar updates in multiple releases (e.g. corrections mapped between the different releases) creating separate module revisions just because the different 3GPP releases use different CR numbers should be avoided. In such case a single new YANG module revision should be created and used in each release. This should contain the CR number from each release.

In order to avoid reusing the same revision date in multiple releases, when a new YANG module revision is needed the revision date shouldbe set as follows. Instead of setting the exact revision date when the module was last edited, the date nearest to that day that is not in the future and that follows the rule below should be used.

When divided by 6, the day in the date should have the same remainder as the release number: (DAY modulo 6 == releaseNumber modulo 6).

Examples:

- Release 17 modulo 6 is 5 ; so day numbers 5, 11, 17, 23, 29 are acceptable while days e.g., 2 or 7 are not.

- Release 18 modulo 6 is 0 ; so day numbers 6, 12, 18, 24, 30 are acceptable while days e.g., 8 or 31 are not.

6.2.1.15 Don’t use YANG statements with their default meaning

YANG statements config, mandatory, max-elements, min-elements, ordered-by, status, yin-element have a specific meaning even if they are absent. The default meaning for these statements should not be explicitly declared in a YANG Module.

E.g. if the mandatory statement is missing that is equivalent to the situation where "mandatory false" is specified; it does not change the meaning of the YANG module, it just makes it longer.

6.2.1.16 Formatting YANG modules/submodules

YANG modules are part of the end-user documentation so to enhance readability the following guidelines should be followed. The guidelines are important as YANG files are often compared and processed as simple text files by SW tools.

- YANG modules should not contain lines longer then 80 characters. (YANG files are often read by the end-users as-is, and reading files with long lines is problematic.)
- A line in a YANG should not contain whitespace (space, tab) immediately before the end of a line or at the end of the file after the last non-blank line. Additional whitespace will confuse tooling when comparing different versions of the YAM.
- Instead of tabs consecutive spaces (a.k.a. soft-tabs) should be used. As different editors use different length tabs (2,4,8 characters long) the indentation of the module might become messed up. Using mixed indentation (both hard-tabs and spaces) is especially problematic.
- In order to avoid long lines the normal indentation should be 2 spaces.
- YANG files should not use characters outside the US-ASCII character set unless there is a specific need for it.
- End-of-line separator SHALL use only a single Newline without a Carriage-Return character.

6.2.1.17 Use original prefix under import statements

The prefix substatement under an import statement shall use the same prefix value, that the imported module declared in it’s prefix substatement under it’s module statement.

6.2.1.18 YANG Naming

All YANG schema nodes and identifiers that are a direct mapping from the stage 2 specifications (including leafs, leaf-list, containers, lists, enumerations, enums, typedefs) shall have the exact same name as used in stage 2 definitions except if

- stage 2 name violates the allowed naming rules of the YANG language as defined in RFC7950 [18] section 6.2.
- Specified otherwise in the present document.

6.2.1.19 Copyright

All YANG modules and submodules shall contain a copyright notice at the end of the module’s/submodule’s description statement.

Standard text is: "Copyright 3GPP Organizational Partners (ARIB, ATIS, CCSA, ETSI, TSDSI, TTA, TTC) <Year>. All rights reserved."

<Year> SHALL be an interval (e.g. 2012-2017) including the year of the file’s creation and last modification or a single 4 digit year if the file was only created/modified in a single year.

Examples:

Copyright 3GPP Organizational Partners (ARIB, ATIS, CCSA, ETSI, TSDSI, TTA, TTC) 2023. All rights reserved.

Copyright 3GPP Organizational Partners (ARIB, ATIS, CCSA, ETSI, TSDSI, TTA, TTC) 2021-2023. All rights reserved.

6.2.2 InformationObjectClass – abstract

6.2.2.1 Introduction

Reference [3] clause 5.4.2

6.2.2.2 YANG mapping

An abstract class shall be mapped to a "grouping". The name of the "grouping" will be <IocName>Grp. The "grouping" shall contain all attributes of the class. The naming attribute shall only be contained as a YANG comment, because all other attributes will be contained in a YANG "non-presence container" named "attributes", however the "key leaf" is contained immediately by the "list", it cannot be inside a child "container".

```
// abstract class MyClass_
grouping MyClass_Grp {
  // contains all contained attributes
  // the leaf of the namingAttribute is either not included or
  // included only as a comment not as a real definition

  // leaf id {
  //   type string;
  //   description "naming attribute of the IOC";
  // }
  leaf attribute1 {..}
  leaf-list attribute2 {..}
                                      }
```

6.2.3 Naming attribute

6.2.3.1 Introduction

Reference [3] clause 3.1

6.2.3.2 Yang mapping

The "leaf" that is mapped from the naming attribute shall be used in the YANG "key" statement. This is usually called "id" as defined in the Top_ class in TS 28.620 Umbrella Information Model (UIM), clause 4.3.8.

6.2.4 InformationObjectClass – concrete

6.2.4.0 Introduction

Reference [3] clause 5.3.2

6.2.4.1 YANG mapping

A concrete class shall be mapped to a "list" that "uses" a "grouping". The "grouping" shall be named <IocName>Grp. It shall contain all attributes of the class in the same manner as the "grouping" for an abstract class. The "list" shall be named <IocName>. The NamingAttribute shall be used as a key. All other attributes shall be placed inside a "container" named "attributes". The "container attributes" will facilitate asking for all attributes of an object instance with a simple subtree or XPath filter.

```
//concrete class
grouping MyConcreteClassGrp {
  // contains all attributes in the same manner as
  //   a grouping for  abstract class
}
```

```
list MyConcreteClass {
  key namingAttribute;  // usually named ‘id’
  leaf namingAttribute {…}
  container attributes {
      uses MyConcreteClassGrp ;
   }
  //YANG lists representing contained classes
}
```

6.2.5 Generalization relationship - inheritance from another class

6.2.5.1 Introduction

Reference [3] clause 5.2.5

Example model: Class `MyManagedFunction` inherits from class `ManagedFunction`.

6.2.5.2 YANG mapping

Generalization/Inheritance relationships are mapped to the inheriting class using the "grouping" of the inherited class in its own "grouping".

```
// Inheritance

grouping ManagedFunctionGrp {
  // Attributes  of ManagedFunction
}

grouping MyManagedFunctionGrp {
  uses ManagedFunctionGrp;
  //additional attributes
}

list MyManagedFunction {
  key id;
  leaf id {}
  container attributes {
    uses MyManagedFunctionGrp;
  }
}
```

6.2.6 Name containment

6.2.6.1 Introduction

Reference [3] clause 5.2.4 - Composite aggregation association relationship

Example model: The classes ParentClass and LocalChildClass are defined in the YANG module _3gpp-ParentClass. ParentClass name-contains LocalChildClass. Another YANG module (_3gpp-ChildClass) defines classes ChildClass1 and ChildClass2. ParentClass name-contains ChildClass1 and ChildClass2.

As on Stage 2 all name-containment is optional, an if-feature statement should be added under “list”, “uses” or “augment” statements modeling name-containment. However, if a YANG module models only a single containment relationship, which is modeled by an augment statement, the if-feature statement is not needed, as the optionality is modeled with the implementation or the non-implementation of the module.

The YANG feature should be named <Child>Under<ParentIocName> . The <Child> section is usually not the name of a specific class, but some name identifying a collection of child classes. The feature statement should be placed in the YANG module where it is used.

Even if a containment relationship (and the contained IOC) is marked as not supported by the YANG feature, any imported but not implemented YANG modules still need to be present in the product with a conformance statement

import-only.(See RFC 8525 [19] conformance-type indicated either by leaf conformance-type or by placing the module under the import-only-module list).. This should not be a problem for implementers as real implementation is not needed, only the YANG files need to be present.

6.2.6.2 YANG mapping

6.2.6.2.1 General

The containment of classes defined in the same YANG module is mapped as embedded "lists".

Containment of classes defined in different YANG modules can be mapped in one of two ways.

6.2.6.2.2 Simple augment

Containment is mapped using the "augment" statement. This is the preferred method.

```
// Class containment
module _3gpp-ParentClass {
  feature LocalChildClassUnderParentClass {
    description “Indicates that LocalChildClass is contained under ParentClass”;
  }

  grouping LocalChildClassGrp {
    // LocalChildClass attributes
  }
  grouping ParentClassGrp {
    // ParentClass attributes
  }

  list ParentClass {
    key id;
    leaf id {}
    attributes {
      use ParentClassGrp;
    }

    list LocalChildClass {
      if-feature LocalChildClassUnderParentClass ;
      key id;
      leaf id {}
      attributes {
        uses LocalChildClassGrp;
      }
    }
    // place to insert/augment child classes
  }
}

module _3gpp-ChildClass {
  import _3gpp-ParentClass { prefix xx3gpp;}

  feature ChildClass1UnderParentClass {
    description “Indicates that ChildClass1 is contained under
      ParentClass”;
  }
  feature ChildClass2UnderParentClass {
    description “Indicates that ChildClass2 is contained under
      ParentClass”;
  }

  grouping ChildClass1Grp {
    // ChildClass1Grp attributes
  }

  grouping ChildClass2Grp {
    // ChildClass2Grp attribute
  }

  augment /xx3gpp:ParentClass {
    if-feature ChildClass1UnderParentClass;
    list ChildClass1 {
```

```
      key id;
      leaf id {}
      attributes {
        uses ChildClass1Grp;
      }
    }
  }
  augment /xx3gpp:ParentClass {
    if-feature ChildClass2UnderParentClass;
    list ChildClass2 {
      key id;
      leaf id {}
      attributes {
        uses ChildClass2Grp;
      }
    }
  }
}
```

6.2.6.2.3 Uses + Subtree grouping

Containment is mapped using the "uses" statement towards a subtree grouping that contains the lists representing the child IOCs; e.g. ParentClass contains ChildClass1 and ChildClass2. This method is recommended when a group of multiple classes is contained together in a number of other classes. In this case optionality is handled on the common subtree level. (The subtree may actually be a group of classes or multiple trees.)

```
// Class containment
module _3gpp-ParentClass {
  import _3gpp-ChildClass { prefix yyy3gpp; }

  feature CommonUnderParentClass {
    description "Indicates that the CommonSubtree shall be contained
      under ParentClass";
  }

  feature LocalChildClassUnderParentClass {
    description “Indicates that LocalChildClass is contained under
      ParentClass”;
  }

  grouping LocalChildClassGrp {
    // LocalChildClass attributes
  }

  grouping ParentClassGrp {
    // ParentClass attributes
  }

  list ParentClass {
    key id;
    leaf id {}
    attributes {
      use ParentClassGrp;
    }
    list LocalChildClass {
      if-feature LocalChildClassUnderParentClass ;
      key id;
      leaf id {}
      attributes {
        uses LocalChildClassGrp;
      }
    }
    uses yyy3gpp:CommonSubtree {
      if-feature CommonUnderParentClass ;
    }
  }
}

module _3gpp-ChildClass {
  grouping ChildClass1Grp {
    // ChildClass1Grp attributes
  }

  grouping ChildClass2Grp {
    // ChildClass2Grp attributes
  }
```

```
  grouping CommonSubtree {
    list ChildClass1 {
      key id;
      leaf id {}
      attributes {
        uses ChildClass1Grp;
      }
    }
    list ChildClass2 {
      key id;
      leaf id {}
      attributes {
        uses ChildClass2Grp;
      }
    }
  }
}
```

6.2.7 Recursive containment - reference based solution

The NRM information object class stage 2 definition contains one case where a class contains itself (so called recursive containment): the It is the `SubNetwork` class.

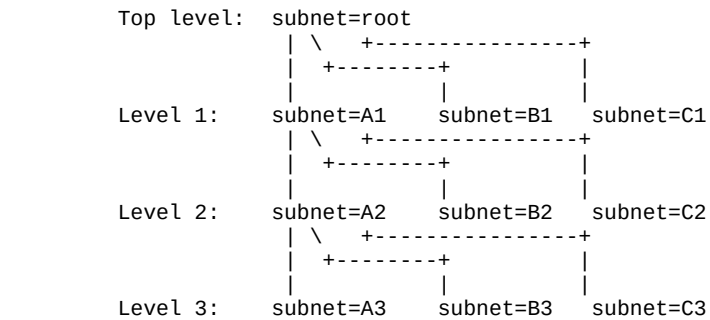
The name containment that a class has with itself in the stage 2 definition shall be modeled using a pair of "leaf-list" references between the instances of the class. The references shall be named "leaf-list parents {…}" and "leaf-list containedChildren {…}". Note the 2 reference "leaf-lists" should be defined directly under the "list" defining the class not in its "grouping" because the "path" statements are specific to each class, so the "leaf-lists" must not be inherited.

```
    list SubNetwork {
      key id;
      leaf id {..}

      container attributes {
        uses SubNetworkGrp;
        leaf-list parents {
          description "Reference to all containg SubNetwork instances
            in strict order from the root subnetwork down to the immediate
            parent subnetwork.
            If subnetworks form a containment hierarchy this is
            modeled using references between the child SubNetwork and the parent
            SubNetworks.
            This reference MUST NOT be present for the top level SubNetwork and
            MUST be present for other SubNetworks.";
          type leafref {
            path "../../../SubNetwork/id";
          }
        }

        leaf-list containedChildren{
          description "Reference to all directly contained SubNetwork instances.
            If subnetworks form a containment hierarchy this is
            modeled using references between the child SubNetwork and the parent
            SubNetwork.";
          type leafref {
            path "../../../SubNetwork/id";
          }
        }
      }
```

The following instance data example shows how the reference values specify the SubNetwork hierarchy:



```
Top level: id=root        parents=null         containedChildren= A1,B1,C1
Level 1:   id=A1,(B1,C1)  parents=root         containedChildren = A2,B2,C2
Level 2:   id=A2,(B2,C2)  parents=root,A1      containedChildren = A3,B3,C3
Level 3:   id=A3,(B3,C3)  parents=root,A1,A2   containedChildren = A4,B4
```

When reading/writing self-contained classes only the last such class instance needs to be specified in the Netconf request as that uniquely identifies the exact instance. The following Netconf request could be used to retrieve all attributes of `SubNetwork`=root, `SubNetwork`=A1, `SubNetwork`=B2, `NRFrequency`=22

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <!-- SubNetwork=root, SubNetwork=A1, SubNetwork=B2, NRFrequency=22 -->
    <filter type="subtree"/>
     <SubNetwork>
       <id>B2</id>
       <NRFrequency>
         <id>22</>
         <attributes/>
       </NRFrequency>
     </SubNetwork>
   </get-config>
</rpc>
```

There is no need to specify the ancestors `SubNetwork`=root, `SubNetwork`=A1 as any `subNetwork` can be addressed directly.

6.2.8 Multi-root management tree

YANG supports multi-rooted managed models natively; the standardized IETF models have many root "list"/"container" nodes.

6.2.9 Alternative containment

Stage 2 models allows multiple different name-containment hierarchies. A particular name-containment hierarchy implemented by a specific vendor/product can be discovered in run-time, by reading the content of the ietf-yang-library and the ietf-yang-schema mount modules.

YANG provides multiple possible methods to model alternative containment hierarchies.

In cases where the number of YANG modules affected by the alternative containment is small, the use of a feature-controlled augmentation is proposed.

```
augment "/SubNetwork" {
  if-feature ExternalsUnderSubNetwork ;
  uses ExternalNRCellCUWrapper;
}
```

In cases where the number of YANG modules affected by the alternative containment is large (cca. more than 8), the following mapping is proposed (using the optional containment of SubNetwork and ManagedElement as an example):

- If the `ManagedElement` is a root class, no further documentation or implementation steps are required.
- If the `ManagedElement` shall be contained under `Subnetwork` it shall be mounted under the `SubNetwork` "list" using the YANG schema mount mechanism as described in RFC 8528 [13].

Mounted schemas will appear in Netconf, the CLI and management GUIs as if they were part of a common containment hierarchy.

Yang Schema Mount provides vendor the flexibility of arranging the containment tree in accordance of operator intention, and provides a way for a consumer to discover the actual mount and containment hierarchy in run-time.

6.2.10 Attribute – simple, single value

6.2.10.1 Introduction

Reference TS 32.156 [3] clause 5.2.1

The multiplicity of the attribute is either 0..1 or 1..1.

6.2.10.2 YANG Mapping

Non-structured single value attributes are mapped to a "leaf".

```
// attribute single value, nonstructured
leaf myAttribute { type xxx; }
```

6.2.11 Attribute – simple, multivalue

6.2.11.1 Introduction

Reference [3] clause 5.2.1

The multiplicity of the attribute may be greater than 1.

6.2.11.2 YANG mapping

If the attribute is isUnique=true it shall be mapped mapped to a leaf-list.

If the attribute is isUnique=false it shall be mapped to a list with an additional dummy index. The name of the list shall be <attributeName>Wrap. The name of the dummyIndex shall be idx and shall have a type uint32 or uint64.

```
// Attribute multivalue, non-structured

// attribute is unique
leaf-list mySimpleMultivalueAttribute1 { type xxx; }

// attribute is non-unique
list mySimpleMultivalueAttribute2Wrap {
  key idx;
  leaf idx { type uint32 ; }
  leaf mySimpleMultivalueAttribute2 {type xxx;}
```

6.2.12 Attribute, structured

6.2.12.0 Introduction

Reference TS 32.156 [3] clause 5.2.1

6.2.12.1 YANG Mapping

Structured attributes are mapped to a grouping containing member parts; and a list using the grouping. (Structured attributes that are not used in multiple places may define the member parts directly in the list.)

```
// attribute, structured, isUnique=true
grouping pLMNIdGrp {
  description "PLMN-Id= Mobile Country Codes (MCC) &
    Mobile Network Codes(MNC)";
  leaf MCC {
     type t_mcc;
   }
```

```
  leaf MNC {
     type t_mnc;
   }
}

list pLMNIdList {
  key "MCC MNC";
  config true;
  description "a list of PLMN-Ids";
  ordered-by user;
  uses pLMNIdGrp;
}


// attribute, structured, isUnique=false
list pLMNIdList {
  key "idx";
  leaf idx { type uint32 ; };
  leaf member1 { type xxx ; }
  leaf member2 { type yyy ; }
}
```

YANG keys for the list shall be selected according to the following steps:

1) If the attribute is isUnique=true and according to the descriptions of the sub-attributes, one or a combination of some subattributes are unique, and all these subattributes are mandatory, these subattribute(s) should be used as key(s) in YANG. (Note only mandatory subattributes should be considered for keys as declaring a subattribute a key makes it mandatory in YANG.)

2) If suitable key(s) cannot be found in step 1, an additional dummy index shall be defined in YANG. The name of the dummyIndex shall be “idx” and shall have a type uint32 or uint64. The dummy key "idx" usually does not appear on stage 2.

6.2.13 defaultValue

6.2.13.1 Introduction

Reference TS 32.156 [3] clause 5.2.1.1.

The 3GPP/UML defaultValue has a different meaning then the YANG "default" statement.

The 3GPP defaultValue could be considered an initialValue as it has effect only at object creation. If the attribute is later deleted the 3GPP defaultValue has no effect. In YANG the "default" is always used whenever a leaf/leaf-list does not have a value: both at creation of the parent object and if the leaf/leaf-list is deleted (set to null in 3GPP operation).

NOTE: Void

The 3GPP defaultValue, isNullable and multiplicity properties cannot be mapped one-to-one into YANG statements. A combination of these three stage 2 input properties shall result in a combination of the four YANG statements mandatory, min-elements, default,and yext3gpp:initial-value (defined in the YANG module _3gpp-common-yang-extensions.yang). The table below describes the combinations of input properties and the resulting YANG statements.

Stage 2 properties			YANG mapping			
multiplicity	isNullable	Stage-2 default	Yang mandatory	YANG Min-Elements > 0	YANG default	YANG initial-value
0..1	True	none	N	N	N	N
		defined	N	N	N	Y
	False	none	N	N	N	N
		defined	N	N	N	Y
1	True	none	N	N	N	N
		defined	N	N	N	Y
	False	none	**Y/N**	Y/N	N	N
		defined	N	N	Y	N
0..*	True	none	N	N	N	N
		defined	N	N	N	Y
	False	none	N	N	N	N
		defined	N	N	N	Y
x..* x >= 1	True	none	N	N	N	N
		defined	N	N	N	Y
	False	none	N	Y	N	N
		defined	N	N	Y	N

YANG mandatory indicates that the leaf shall have a "`mandatory true;`" substatement.

YANG min-elements > 0 indicates that the list or leaf-list shall have a "`min-elements`" substatement that has an argument that is greater than zero.

YANG default indicates that the leaf shall have a "`default`" substatement.

YANG initial-value indicates that the leaf should have a "`yext3gpp:initial-value`" substatement.

6.2.13.2 YANG mapping

YANG "default" and "initial-value" statements are only used for simple attributes. For structured attributes describe the default in the YANG description. In some cases, the stage 2 default value is not defined as a specific value, but rather as a reference or defined in a human readable language. In these cases, the default value is described in the YANG description.

YANG default or yext3gpp:initial-value statements shall be used as specified in the table in clause 6.2.13.1.

NOTE 1: Void

NOTE 2: The YANG extension statement yext3gpp:initial-value is not understood or enforced by standard YANG tools, it needs extra SW implementation.

6.2.14 multiplicity and cardinality

6.2.14.0 Introduction

Reference TS 32.156 [3] clause 5.2.1.1

Reference TS 32.156 [3] clause 5.2.8

6.2.14.1 YANG mapping

YANG mandatory, or min-elements statements shall be used as specified in the table in clause 6.2.13.1.

Multiplicity of attributes mapped to a list or leaf-list shall be mapped to the "min-elements" and "max-elements" YANG statements.

Cardinality for containment of classes shall be mapped to "min-elements" and "max-elements" on the list representing the child objects.

Cardinality for reference relationships shall be mapped to "mandatory", "min-elements" and "max-elements" on the reference attributes representing the reference.

6.2.15 isNullable

6.2.15.0 Introduction

Reference TS 32.156 [3] clause 5.2.1.1

6.2.15.1 YANG mapping

isNullable=false for attributes is not mapped to YANG. In this case the attribute's multiplicity will dictate any YANG mandatory or min-elements statements. See table in clause 6.2.13.1.

isNullable=true shall not be mapped to YANG, because isNullable=true makes the attribute optional to use, which is the default case in YANG, thus it should not be explicitly stated.

A special case is an attribute that is mapped to a list or leaf-lists, is isNullable=true and has a minimum multiplicity greater than zero. In this case a "must" statement shall be added to the list/leaf-list forbidding any multiplicity values between 1 and the minimum multiplicity (but allowing zero and the minimum). See example below:

```
list nullableListWithMinimumMultiplicityOf5 {
  key idx;
  must 'count(.) = 0 or count(.) >= 5';
  leaf idx { type uint32 ; }
  leaf nonUniqueSingleValueAttribute [ type int32; };
```

NOTE: Void

6.2.16 dataType

6.2.16.0 Introduction

Reference TS 32.156 [3] clause 5.3.4

Reference TS 32.156 [3] clause 5.4.3

6.2.16.1 YANG mapping

Mapping for predefined datatypes shall be the following:

- integer -> One of the 8 YANG integer types
- string - > string
- Boolean -> Boolean

3GPP user-defined simple datatypes shall be mapped to the YANG "typedef" statement.

3GPP user-defined structured datatypes shall be mapped to the YANG "grouping" statement with the name <typeName>Grp.

6.2.17 enumeration

6.2.17.0 Introduction

Reference TS 32.156 [3] clause 5.3.5

6.2.17.1 YANG mapping

The 3GPP enumeration datatype shall be mapped to the YANG "enumeration" YANG type.

6.2.18 choice

6.2.18.0 Introduction

Reference TS 32.156 [3] clause 5.3.6

6.2.18.1 YANG mapping

The 3GPP choice stereotype shall be mapped to a Yang "choice" statement.

6.2.19 isInvariant on attribute

Reference [TS 32.156 [3] Model repertoire] clause 5.2.1.1

6.2.19.1 YANG mapping

Attributes with the property isInvariant=true shall be marked with the "yext3gpp:inVariant" extension defined in the YANG module _3gpp-common-yang-extensions.yang in 3GPP TS 28.623[20].

6.2.20 isReadable/isWritable

Reference [TS 32.156 [3] Model repertoire] clause 5.2.1.1

6.2.20.1 YANG mapping

isReadable=false attributes can not be represented in YANG. Assumed not to be a problem. A YANG extension could be defined to handle it if needed.

Attributes with the properties isReadable=true AND isWritable=false shall be mapped to YANG config=false leafs/leaf-lists/lists. As config=false is inherited down the containment tree, it should not be placed on each leaf, leaf-list, etc. once the containing list/container is marked config false;

Attributes with the properties isReadable=true AND isWritable=true shall be mapped to YANG config=true leafs/leaf-lists/lists. "config true;" should not be explicitly declared as that is the default case.

6.2.21 isOrdered

Reference [TS 32.156 [3] Model repertoire] clause 5.2.1.1

6.2.21.1 YANG mapping

For isWritable=true attributes the property isOrdered=true shall be mapped to the "ordered-by user;" YANG statement. For isWritable=false attributes the isOrdered property shall be described in the description statement of the YANG leaf-list, list representing the attribute.

NOTE: The "ordered-by user" statement is ignored in YANG if the leaf-list or list is config=false.

6.2.22 isUnique

Reference [TS 32.156 [3] Model repertoire] clause 5.2.1.1

6.2.22.1 YANG mapping

The property isUnique=True shall be mapped to the YANG "unique" statement. Leaf-list are always unique in YANG, no marking needed.

6.2.23 allowedValues

Reference [TS 32.156 [3] Model repertoire] clause 5.2.1.1

6.2.23.1 YANG mapping

For attributes with a type=integer or a user-defined type based on integers allowedValues shall be mapped to a YANG "range" statement with specific values.

For attributes with a type=string or a user-defined type based on string allowedValues shall be mapped either to an enumerated YANG type or to a sting with alternatives defined using the YANG "pattern" statement.

For attributes with a type=enumeration or a user-defined type based on enumeration allowedValues shall be mapped to a YANG enumeration type restricted with YANG "enum" substatements. (https://tools.ietf.org/html/rfc7950#section-9.6.3)

6.2.24 Xor constraint

Reference [TS 32.156 [3] Model repertoire] clause 5.2.10

6.2.24.1 YANG mapping

Model elements with a Xor constraint shall be mapped to the YANG "choice" statement.

6.2.25 ProxyClass

Reference [TS 32.156 [3] Model repertoire] clause 5.3.1

6.2.25.1 YANG mapping

A proxyclass is not directly mapped to YANG. A proxyclass represents a number of specific classes. Attributes, links, methods (or operations), and interactions that are present in the proxyclass shall be modelled in the represented specific classes.

6.2.26 SupportQualifier

6.2.26.1 Introduction

Reference [3] clause 6 - Qualifiers

6.2.26.2 YANG mapping

SupportQualifier=M is the default case in YANG so it needs no mapping.

SupportQualifier=O shall be mapped the same way as SupportQualifier=M. Just like in the other solution sets the supportQualifier shall not be directly visible in the 3GPP Stage 3 YANG model. The support is indicated the following way:

- If the vendor supports an optional item, there is no further modeling needed
- If the vendor does not support the optional item, it needs to create a separate vendor specific YANG module and include a “deviation” statement in it formally declaring the non-supported parts. A single YANG module may contain any number of deviations. E.g.:

```
deviation /ManagedElement/attributes/optionalAttribute {deviate not-supported;}
```

SupportQualifier=CO {if the item is not supported) is mapped the same way as a not supported SupportQualifier=O item.

SupportQualifier=CM & CO (if item is supported) shall be mapped as a SupportQualifier=M item, also considering the following:

- if the condition can be expressed with XPATH, an additional "when" statement shall be used.
- otherwise make the data node non-mandatory and define the condition in the description statement.

6.2.27 isNotifyable

6.2.27.1 Introduction

Reference TS 32.156 [3] clause 5.2.1.1

6.2.27.2 YANG mapping

Attributes that are isNotifyable=False shall be marked with the "yext3gpp:notNotifiable" YANG extension statement defined in the YANG module _3gpp-common-yang-extensions.yang.

Attributes that are isNotifyable=True shall not be marked in any way, as it is a default case.

6.2.28 LifecycleStatus

6.2.28.1 Introduction

Reference [3] clause 5.2.A - LifecycleStatus

6.2.28.2 YANG mapping

LifecycleStatus=current is the default case in YANG so it needs no mapping.

LifecycleStatus=deprecated shall be mapped to the YANG statement

```
status deprecated;
```

under the relevant leaf, leaf-list, list, container or grouping.

Annex A (informative): Example usage of the template for one management capability

4 Management capabilities

4.1 Lifecycle management

4.1.1 Description

The lifecycle management of the edge components is to be enabled by the 3GPP Management System. The lifecycle management includes instantiation, termination, modification and query of the edge components.

4.1.2 Use cases

4.1.2.1 EAS deployment UC-LM-01

The goal of this use case is to enable ASP to deploy the EAS in the EDN, by requesting the provisioning MnS producer with the deployment requirements (e.g. the topological or geographical service areas, software image information, QoS, affinity/anti-affinity with other EAS, etc.) to deploy the EAS. The provisioning MnS producer returns a response indicating the operation is in progress to prevent the consumer from waiting, as the deployment in the edge cloud may take a while. Since, there can be multiple Edge Data Network (EDN) present/serving a particular edge location. This makes it critical for application service provider to have their EAS deployed at appropriate EDN(s) to provide high performance services for the UE. Therefore, provisioning MnS producer analyses the deployment requirements to determine where (i.e. on which EDN) and how many EAS VNF instance(s) should be instantiated, and requests the NFVO in ETSI NFV MANO to instantiate the EAS VNF instance(s). The provisioning MnS producer sends a notification to ASP indicating the result of instantiation (e.g. success, failure) when a notification is received from NFVO indicating the result of instantiation operation

4.1.3 Requirements

Requirement label	Description	Related use case(s)
REQ-EAS-INST-FUN-1	Generic provisioning MnS producer should have a capability allowing an authorized consumer to request the deployment of EAS based on the given deployment requirements.	UC-LM-01

Annex B (informative): Change history

Change history							
Date	**Meeting**	**TDoc**	**CR**	**Rev**	**Cat**	**Subject/Comment**	**New version**
2019-09	SA#85					Change control version	16.0.0
2019-12	SA#86	SP-190172	0001	-	F	Implement Edithelp comments	16.1.0
2019-12	SA#86	SP-190172	0002	-	F	Solutions for Editor's notes →Not implemented due to CR clash (MCC)	16.1.0
2019-12	SA#86	SP-190172	0003	1	F	Resolution of Editors Note in clause W4.3 Class definitions→ not implemented due to CR clash (MCC)	16.1.0
2019-12	SA#86	SP-191166	0004	2	B	Additions to YANG style Guide	16.1.0
2020-03	SA#87E	SP-200169	0005	-	B	YANG Guidelines Update	16.2.0
2020-03	SA#87E	SP-200172	0006	-	F	Remove incorrect example from constraints table	16.2.0
2020-03	SA#87E	SP-200172	0007	-	F	Resolution of Editors Note in clause W4.3 Class definitions	16.2.0
2020-07	SA#88E	SP-200489	0008	1	B	Update YANG Guidelines	16.3.0
2020-07	SA#88E	SP-200490	0009	-	B	Update YANG Guidelines	16.3.0
2020-12	SA#90e	SP-201089	0011	-	F	Fix incorrect reference in notification template	16.4.0
2020-12	SA#90e	SP-201089	0012		F	Update thresholdCrossingNotification to be a common notification.	16.4.0
2020-12	SA#90e	SP-201052	0010	1	B	Import prefix rule for YANG	17.0.0
2021-03	SA#91e	SP-210155	0015	-	C	YANG containment mapping	17.1.0
2021-03	SA#91e	SP-210155	0016	-	C	Code begin end markers and longer prefix length	17.1.0
2021-06	SA#92e	SP-210407	0013	4	C	Update on template for requirement specifications	17.2.0
2021-09	SA#93e	SP-210878	0021	1	A	Align different (abbreviated) names for support qualifier to “S”	17.3.0
2021-09	SA#93e	SP-210887	0022	-	F	Change format for NRM stage 3 definition rules from JSON to YAML	17.3.0
2021-09	SA#93e	SP-210887	0023	-	B	Add motivation to requirements	17.3.0
2022-06	SA#96	SP-220509	0027	-	F	YANG Mapping Corrections	17.4.0
2022-09	SA#97e	SP-220859	0028	-	F	YANG Mapping Rule Corrections	17.5.0
2022-12	SA#98e	SP-221171	0030	-	A	Add missing mapping of isNotifyable	17.6.0
2022-12	SA#98e	SP-221189	0031	-	B	Deprecating model elements	18.0.0
2023-03	SA#99	SP-230212	0032	1	C	Clarifications for lifecycleStatus property	18.1.0
2023-03	SA#99	SP-230197	0034	-	A	Correction of RFC reference	18.1.0
2023-03	SA#99	SP-230201	0035	2	F	Clarification of Requirements and Use Case template	18.1.0
2023-06	SA#100	SP-23650	0037	-	A	Allow YANG anydata	18.2.0
2023-06	SA#100	SP-230648	0040	-	A	Correction to template for NRM description related to Notifications	18.2.0
2023-09	SA#101	SP-230945	0042	-	A	Include copyright in YANG Files R18	18.3.0
2023-12	SA#102	SP-231492	0044	1	A	Rel18 CR 32.160 Correct YANG mapping of isInvariant	18.4.0
2023-12	SA#102	SP-231492	0046	1	A	Rel-18 CR 32.160 Clarify YANG Vendor extensions	18.4.0
2023-12	SA#102	SP-231492	0048	-	A	Rel-18 CR 32.160 Clarify YANG revision date handling	18.4.0
2024-03	SA#103	SP-240185	0050	-	A	Rel-18 CR 32.160 Clarify YANG revisions	18.5.0

History

Document history		
V18.5.0	May 2024	Publication