International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

X.785
(07/2021)

SERIES X: DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY

OSI management – Management functions and ODMA functions

Guidelines for defining REST-based managed objects and management interfaces

Recommendation ITU-T X.785

ITU-T X-SERIES RECOMMENDATIONS

DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY

PUBLIC DATA NETWORKS	
Services and facilities	X.1–X.19
Interfaces	X.20–X.49
Transmission, signalling and switching	X.50–X.89
Network aspects	X.90–X.149
Maintenance	X.150–X.179
Administrative arrangements	X.180–X.199
OPEN SYSTEMS INTERCONNECTION	
Model and notation	X.200–X.209
Service definitions	X.210–X.219
Connection-mode protocol specifications	X.220–X.229
Connectionless-mode protocol specifications	X.230–X.239
PICS proformas	X.240–X.259
Protocol Identification	X.260–X.269
Security Protocols	X.270–X.279
Layer Managed Objects	X.280–X.289
Conformance testing	X.290–X.299
INTERWORKING BETWEEN NETWORKS	
General	X.300–X.349
Satellite data transmission systems	X.350–X.369
IP-based networks	X.370–X.379
MESSAGE HANDLING SYSTEMS	X.400–X.499
DIRECTORY	X.500–X.599
OSI NETWORKING AND SYSTEM ASPECTS	
Networking	X.600–X.629
Efficiency	X.630–X.639
Quality of service	X.640–X.649
Naming, Addressing and Registration	X.650–X.679
Abstract Syntax Notation One (ASN.1)	X.680–X.699
OSI MANAGEMENT	
Systems management framework and architecture	X.700–X.709
Management communication service and protocol	X.710–X.719
Structure of management information	X.720–X.729
Management functions and ODMA functions	**X.730–X.799**
SECURITY	X.800–X.849
OSI APPLICATIONS	
Commitment, concurrency and recovery	X.850–X.859
Transaction processing	X.860–X.879
Remote operations	X.880–X.889
Generic applications of ASN.1	X.890–X.899
OPEN DISTRIBUTED PROCESSING	X.900–X.999
INFORMATION AND NETWORK SECURITY	X.1000–X.1099
SECURE APPLICATIONS AND SERVICES (1)	X.1100–X.1199
CYBERSPACE SECURITY	X.1200–X.1299
SECURE APPLICATIONS AND SERVICES (2)	X.1300–X.1499
CYBERSECURITY INFORMATION EXCHANGE	X.1500–X.1599
CLOUD COMPUTING SECURITY	X.1600–X.1699
QUANTUM COMMUNICATION	X.1700–X.1729
DATA SECURITY	X.1750–X.1799
IMT-T SECURITY	X.1800–X.1819

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T X.785

Guidelines for defining REST-based managed objects and management interfaces

Summary

Recommendation ITU-T X.785 defines a set of guidelines for managed object modelling and a management interface for representational state transfer (REST)-based network management. It is part of a framework for REST-based network management interfaces. It specifies how REST-based management interfaces should be defined. It covers the generic accessing methods of REST-based managed objects, accessing methods for specific managed object (Mos), information modelling in REST / hypertext transfer protocol (HTTP) and YAML ain't markup language (YAML) / JavaScript object notation (JSON) schemas. Some HTTP requests/responses and YAML/JSON schemas are provided for defining some basic data types: generic managed object (MO) and generic MO accessing methods.

History

Edition	Recommendation	Approval	Study Group	Unique ID*
1.0	ITU-T X.785	2021-07-29	2	11.1002/1000/14745

Keywords

REST, framework, guidelines, network management.

* To access the Recommendation, type the URL http://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID. For example, http://handle.itu.int/11.1002/1000/11830-en.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents/software copyrights, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the appropriate ITU-T databases available via the ITU-T website at http://www.itu.int/ITU-T/ipr/.

© ITU 2021

All rights reserved. No part of this publication may be reproduced, by any means whatsoever, without the prior written permission of ITU.

Table of Contents

Page

1 Scope ... 1
2 References ... 1
3 Definitions ... 2
3.1 Terms defined elsewhere ... 2
3.2 Terms defined in this Recommendation ... 2
4 Abbreviations and acronyms ... 2
5 Conventions ... 3
6 Overview of a REST-based management framework ... 4
6.1 Overview ... 4
6.2 Resources ... 4
6.3 Definition languages of REST-based interface ... 5
7 Principles for REST-based interface design ... 5
8 Definition of a generic managed object using YAML schema ... 6
8.1 REST role in management interfaces ... 6
8.2 Definition of managed objects using JSON/YAML schema ... 7
9 Accessing methods for managed objects ... 12
9.1 Generic MO accessing methods ... 12
9.2 Design guidelines for specific MO class accessing methods ... 13
10 Information modelling guidelines for REST-based interfaces ... 15
10.1 Resource Modeling ... 15
10.2 Attribute ... 16
10.3 Name conventions for MOCs, attributes and data types ... 16
10.4 Other guidelines ... 16
11 Compliance and conformance ... 16
11.1 Standards document compliance ... 16
11.2 System conformance ... 17
11.3 Conformance statement guidelines ... 17
Annex A – Common REST-based YAML/JSON schema definitions ... 18
A.1 YAML schema definitions for the generic managed object and common data types ... 18
A.2 YAML/JSON schema definitions for common object accessing methods ... 21
Appendix I – An example of REST-based interface definitions for resource ... 38
I.1 An example showing the CRUD definitions for a specific resource ... 38
Appendix II – Usage examples of the ContainmentRelationshipType and AssociationRelationType ... 44
Appendix III – Background for REST and HTTP technologies ... 46
III.1 Background ... 46
III.2 Short review of REST and HTTP ... 46

Page

III.3 Benefits of introducing REST into network management domain 48

Bibliography.. 49

Recommendation ITU-T X.785

Guidelines for defining REST-based managed objects and management interfaces

1 Scope

The network management architecture defined in [ITU-T M.3010] introduces the use of multiple management protocols. So far, the guidelines for the definition of managed objects (GDMO) / common management information protocol (CMIP), common object request broker architecture (CORBA) / Internet inter-ORB protocol (IIOP), structure of management information (SMI) / simple network management protocol (SNMP), web services / simple object access protocol (SOAP) are possible choices at the application layer. Based on the management interface specification methodology defined in [ITU-T M.3020], more technology-based paradigms can be introduced into network management interfaces, and REST/HTTP is now an additional paradigm for network management.

This Recommendation sets out a framework for defining how interfaces supported by management systems and network elements should be modelled using JSON/YAML schemas. It is within the scope of this Recommendation to provide the following guidelines or instructions:

– principles for REST interface designs;
– containment, association and inheritance relationship and naming rules for managed entities;
– generic accessing methods for managed objects;
– guidelines for defining accessing methods for specific resources;
– information modelling guidelines for REST-based interfaces;
– common data type definitions.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T M.3010] Recommendation ITU-T M.3010 (2000), *Principles for a telecommunications management network*.

[ITU-T M.3020] Recommendation ITU-T M.3020 (2017), *Management interface specification methodology*.

[ITU-T M.3160] Recommendation ITU-T M.3160 (2008), *Generic, protocol-neutral management information model*.

[ITU-T M.3701] Recommendation ITU-T M.3701 (2010), *Common management services – State management – Protocol neutral requirements and analysis*.

[ITU-T X.701] Recommendation ITU-T X.701 (1997), *Information technology – Open Systems Interconnection – Systems management overview*.

[ITU-T X.703] Recommendation ITU-T X.703 (1997), *Information technology – Open Distributed Management Architecture*.

[RFC 3986] IETF RFC 3986 (2005), *Uniform Resource Identifier (URI): Generic Syntax*.

[RFC 5789] IETF RFC 5789 (2010), *PATCH Method for HTTP*.

[RFC 6585] IETF RFC 6585 (2012), *Additional HTTP Status Codes*.

[RFC 6901] IETF RFC 6901 (2013), *JavaScript Object Notation (JSON) Pointer*.

[RFC 6902] IETF RFC 6902 (2013), *JavaScript Object Notation (JSON) Patch*.

[RFC 7230] IETF RFC 7230 (2014), *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*.

[RFC 7231] IETF RFC 7231 (2014), *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*.

[RFC 7232] IETF RFC 7232 (2014), *Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests*.

[RFC 7396] IETF RFC 7396 (2014), *JSON Merge Patch*.

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

3.1.1 agent [ITU-T M.3020]: Encapsulates a well-defined subset of management functionality. It interacts with managers using a management interface. From the manager's perspective, the agent behaviour is only visible via the management interface.

3.1.2 managed object class [ITU-T X.701]: A named set of managed objects sharing the same (named) sets of attributes, notifications, management operations (packages), and which share the same conditions for presence of those packages.

3.1.3 manager [ITU-T M.3020]: Models a user of agent(s) and it interacts directly with the agent(s) using management interfaces. Since the manager represents an agent user, it gives a clear picture of what the agent is supposed to do. From the agent perspective, the manager behaviour is only visible via the management interface.

3.1.4 notification [ITU-T X.703]: An interaction for which the contract between the invoking object (client) and the receiving object (server) is restricted to the ability of the server to receive the contents of information sent by the client.

3.2 Terms defined in this Recommendation

This Recommendation does not define any new terms.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

API Application Programming Interface

BNF Backus-Naur Form

CMIP Common Management Information Protocol

CORBA Common Object Request Broker Architecture

CRUD Create, Retrieve, Update, Delete

DN Distinguished Name

GDMO	Guidelines for the Definition of Managed Objects
HTTP	Hypertext Transfer Protocol
IIOP	Internet Inter-ORB Protocol
IT	Information Technology
JSON	JavaScript Object Notation
MO	Managed Object
MOC	Managed Object Class
MOI	Managed Object Instance
OAS	OpenAPI Specification
OSI	Open System Interconnection
RDN	Relative Distinguished Name
REST	Representational State Transfer
RPC	Remote Procedure Call
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
TMN	Telecommunications Management Network
URI	Unified Resource Identifier
XML	extensible Markup Language
YAML	YAML Ain't Markup Language

5 Conventions

A few conventions are followed in this Recommendation to make the reader aware of the purpose of the text. While most of the Recommendation is normative, paragraphs succinctly stating mandatory requirements to be met by a management system (managing and/or managed) are preceded by a boldface "R" enclosed in parentheses, followed by a short name indicating the subject of the requirement and a number. For example:

(R) EXAMPLE-1 An example mandatory requirement.

Requirements that may be optionally implemented by a management system are preceded by an "O" instead of an "R". For example:

(O) EXAMPLE-2 An example optional requirement.

The requirement statements are used to create compliance and conformance profiles.

Examples of JSON and YAML schemas are included in this Recommendation and normative JSON or YAML schema specifying the data types, base classes and other modelling constructs of the framework are included in Annex A. The JSON/YAML schemas are written in a 10 point courier typeface:

<table>
<tr><td>A JSON schema example</td><td><pre>{
    "title": "root",
    "items": {
        "title": "array item"
    }
}</pre></td></tr>
<tr><td>A YAML schema example</td><td><pre>SomeType:
  type: object
    properties:
      attr1:
        type: string
      attr2:
        type: string
        enum:
          - e1
          - e2</pre></td></tr>
</table>

6 Overview of a REST-based management framework

6.1 Overview

REST-based technologies have been widely used in the information technology (IT) industry. Appendix III provides more information on the features of REST technology. REST technology is similar to web services technology and can be used in network management interfaces. The REST technology uses a resource-oriented approach to define information entities, and unified resource identifiers (URIs) ([RFC 3986]) for entity identification, and the corresponding operations are also defined with a close relation with the resource URIs.

This Recommendation sets up a framework for defining how interfaces supported by management systems and network elements should be modelled using REST application programming interface (API) and JSON/YAML schemas (see [RFC 6901], [RFC 6902], and [b-OAI-OAS3]).

The complete REST-based management framework includes the following aspects:

1) Managed objects and interface definition guidelines:
   - definition of managed object classes using YAML schema;
   - inheritance, containment and association relationships of managed objects (Mos);
   - accessing methods for managed object instances (MOIs);
   - information modelling guidelines for REST-based interfaces;
2) REST-based supporting services for network management:
   - definition of a REST-based notification service;
   - definition of a REST-based heartbeat service;
   - definition of a REST-based containment service.

This Recommendation mainly deals with managed objects and interface definition guidelines. REST-based supporting services will be dealt with in other Recommendations.

6.2 Resources

Hypertext transfer protocol (HTTP) ([RFC 7230]) uses a different terminology based on the notion of resources, as defined in clause 2 of [RFC 7231]. Each resource is represented by a resource representation as defined in clause 3 of [RFC 7231]. Valid resource representations are e.g. extensible markup language (XML) instance documents or JSON instance documents.

Resources can be classified according to their structure and behaviour into resource archetypes. This helps to specify clear and understandable interfaces. The following three archetypes are defined (also aligned with [b-3GPP TS 32.158]):

– **Document resource**: This is the standard resource containing data in form of name-value pairs and links to related resources. This kind of resource typically represents a real-world object or a logical concept.
– **Collection resource**: A collection resource is grouping resources of the same kind. The resources below the collection resource are called items of the collection. An item of a collection is normally a document resource. Collection resources typically contain links to the items of the collection and information about the collection like the total number of items in the collection. Collection resources can be further distinguished into server-managed and client-managed resources. Collection resources are also known as container resources.
– **Operation resource**: Operation resources represent executable functions. They may have input and output parameters. Operation resources allow some kind of fallback to a remote procedure call (RPC) style design in case application specific actions cannot be mapped easily to create, retrieve, update, delete (CRUD) style operations.

6.3 Definition languages of REST-based interface

This Recommendation follows the OpenAPI specification (OAS, see [b-OAI-OAS3]) to define REST-based interfaces. The OpenAPI specification (OAS) is developed by the OpenAPI initiative, which defines a standard, programming language-agnostic interface description for REST APIs, which allows both humans and computers to discover and understand the capabilities of a service without requiring access to a source code, additional documentation, or inspection of network traffic. There are two languages that can be used in OAS: JSON and YAML, and the relationship between them is described below.

YAML is a human-friendly, cross language, unicode based data serialization language designed around the common native data types of agile programming languages. It is broadly useful for programming needs ranging from configuration files to internet messaging to object persistence to data auditing.

Both JSON and YAML aim to be human readable data interchange formats. However, JSON and YAML have different priorities. JSON's foremost design goal is simplicity and universality. Thus, JSON is trivial to generate and parse, at the cost of reduced human readability. It also uses a lowest common denominator information model, ensuring any JSON data can be easily processed by every modern programming environment.

In contrast, YAML's foremost design goals are human readability and support for serializing arbitrary native data structures. Thus, YAML allows extremely readable files but is more complex to generate and parse. In addition, YAML ventures beyond the lowest common denominator data types, requiring more complex processing when crossing between different programming environments.

YAML can therefore be viewed as a natural superset of JSON, offering improved human readability and a more complete information model. This is also the case in practice; every JSON file is also a valid YAML file. This makes it easy to migrate from JSON to YAML if/when the additional features are required.

Within this Recommendation, both JSON and YAML schemas will be used for interface definitions.

7 Principles for REST-based interface design

This clause identifies some interface design considerations that should be addressed by this framework through REST interfaces. It provides the modelling principles for REST-based managed objects (MO) and their accessing methods.

The REST-based design considerations related to REST APIs and JSON/YAML schemas and modelling concerns super-classes, naming of managed objects and resource-oriented interfaces, operations and notifications.

This Recommendation defines a lightweight generic use of REST-based interface design patterns. The management and controlling functions are defined using HTTP methods, not just for an individual management object class.

The framework has the following principles to define a REST-based management information model and interfaces.

– All interface interactions are defined as HTTP methods, each operation includes a request and an optional corresponding response when needed.
– Each managed object class (MOC) is defined as a resource when exchanged through the management interface, and each attribute or state of the MOC is defined as a property in the resource.
– The naming of MOC instances follows the concept of a URI, which can be accessed using an HTTP request.
– There are four basic accessing methods for managed objects in the traditional telecommunications management network (TMN) management paradigm, which are: createMO, deleteMO, getMOAttributes, and setMOAttribute. These methods are redefined in this management framework using HTTP POST, HTTP DELETE, HTTP GET, and HTTP PUT/PATCH. These methods are applicable for every MOC instances, and the URI is used to indicate which instances are accessed using these methods.
– Other interface control functions are defined as HTTP POST methods against a specific resource.
– Common data types are defined in JSON/YAML schemas which can be shared by application-specific interface definitions.

8 Definition of a generic managed object using YAML schema

8.1 REST role in management interfaces

To support the software objects representing manageable resources, a base class is defined for use in modelling network resources. Other MOCs (managed object class) in information models must be derived from this base class in order to operate within this framework. Some generic accessing methods and some other extended functions are defined to provide interfaces to manage MOs.

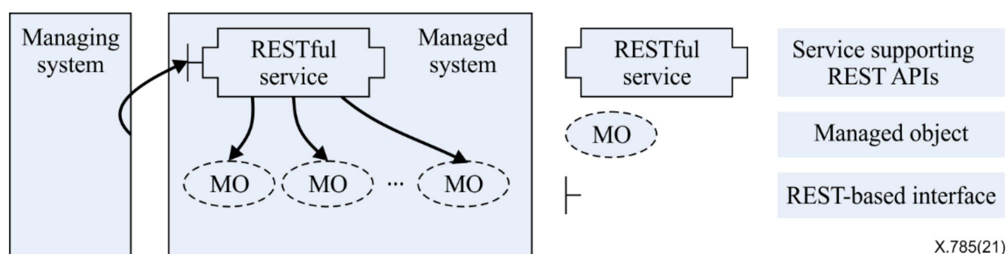

Figure 1 – RESTful services role

Figure 1 shows how a managing system accesses a managed system that supports a RESTful services interface. A RESTful services interface acts as an intermediate entity that enables a managing system to manage proper MOs in a managed system representing manageable resources.

8.2 Definition of managed objects using JSON/YAML schema

An MO is an open system interconnection (OSI) management view of a resource that is subject to management, such as a connection or an item of physical equipment. It is the abstraction of such a resource that represents its properties for the purpose of management. An MO may also include attributes that provide information used to characterize itself and operations that represent its behaviours. The purpose of the framework is to provide a collection of capabilities to manage these MOs. MOs need some approaches to describe their properties and behaviours. In REST-based technology, an MO is a managed entity that represents a manageable resource in terms of shared state and behaviour where state and behaviour are separated through outsourcing of the behaviour to an assigned so-called "managing entity" (e.g., a service and its interface) that takes a steward role with regard to the behaviours of its allocated managed entities. Since an MO's state and behaviour can be separated, the state can be described by JSON/YAML schema and behaviour by REST APIs. One important benefit of using a JSON/YAML document to store an MO's state is that REST APIs can also use JSON/YAML schema to describe the data type of its exchanged messages, and these JSON/YAML based MOs' information can be exchanged without any modification.

8.2.1 Definition of a generic managed object class

A managed object class is a further abstraction of managed objects. All network resources have some common attributes and all MOCs shall inherit, either directly or indirectly, from a superclass, namely a ManagedObject_C (the suffix "_C" indicates it represents an MOC, not just a data type). Using ManagedObject_C to define new MOCs will be easier and faster and provide better maintenance. As mentioned above, all MOCs are described in JSON/YAML schema and the data type of ManagedObject_C is given in Table 1 and the attributes can be found in Table 2.

Table 1 – Data type of superclass ManagedObject_C

```
ManagedObject_C:
  type: object
  required:
    - objectClass
    - objectInstance
    properties:
      objectClass:
        type: string
      objectInstance:
        type: string
        format: uri
      creationSource:
        $ref: '#/components/schemas/SourceIndicatorType'
```

```
SourceIndicatorType:
   type string
   enum:
     - resourceOperation
     - managementOperation
     - unknown
```

Table 2 – Attributes of superclass ManagedObject_C

Attribute name	Support qualifier	Read qualifier	Write qualifier
objectClass	Mandatory	Mandatory	–
objectInstance	Mandatory	Mandatory	–
creationSource	Optional	Mandatory	–

As shown in Table 2, ManagedObject_C is made up of three attributes including objectClass, objectInstance, and creationSource. An attribute has an associated value with a specific data type. The attribute objectClass is used to identify the class type of this MO instance. The attribute objectInstance is used to uniquely identify an MO instance, and the data type is a string with the format of URI, which will be further explained in formula (1). The attribute creationSource indicates whether an MO is created automatically in a managed system, or by a managing system through a management operation, or unknown.

In network management, each MOI is uniquely identified by the object instance name. Considering the REST feature, each managed object can be regarded as a resource, and the URI as the unique identifier of the resource can naturally be the only instance of the managed object class. Resources in REST include document resources, collection resources, and task resources. The object instance is named URI string conforming to the following Backus-Naur form (BNF) paradigm specification, as shown in Table 3.

Table 3 – BNF paradigm specification for URI

```
URI = {URI-prefix}/{ResourcePath}
URI-prefix = {irpRoot}/{irpName}/{irpVersion}
ResourcePath={DocumentResourcePath}|{CollectionResourcePath}|{TaskResourcePath}
DocumentResourcePath = ( "/" {RDN} )+
CollectionResourcePath=("/" {RDN})+ "/" {className}
TaskResourcePath=("/"{RDN})+ ["/" {className}] "/" {actionName}
RDN={className} "=" {namingAttributeValue}
className = The class name of the specific MO instance.
namingAttributeValue = The value of the naming attribute of the specified MO instance.
```

(1)

The namingAttributeValue in the above formula should be the actual value of the naming attribute of an MOI.

A complete YAML schema definition for the generic ManagedObject_C is defined in Annex A.1.

(R) OBJECT-1. All the classes used to model resources on a managed system shall inherit (directly or indirectly) from the ManagedObject_C described above and defined in the YAML schema in clause A.1. The capabilities described above shall be supported.

8.2.2 Inheritance relationship of managed objects

Inheritance is an important concept in the object-oriented mechanism. When defining a new object class, in order to reuse the definition of the existing object class, some or all of the features defined in the existing object class can be inherited as the characteristics of the new object class. New object classes can also define additional features. In network management, all manageable managed object classes inherit directly or indirectly from the ManagedObject_C base class, and extend their own unique attribute definitions based on public attributes to more clearly express their own feature information. In YAML Schema, the inheritance of an attribute is represented by "allOf".

Taking a managed element in a telecommunication network as an example, a managed element may be composed of multiple frames, each of which includes several racks, and multiple slots are included in a rack, and circuit packs performing various functions are inserted into the slots. According to the inheritance relationship in [ITU-T M.3160], the related object classes and the inheritance relationship between each object class are shown in Figure 2.

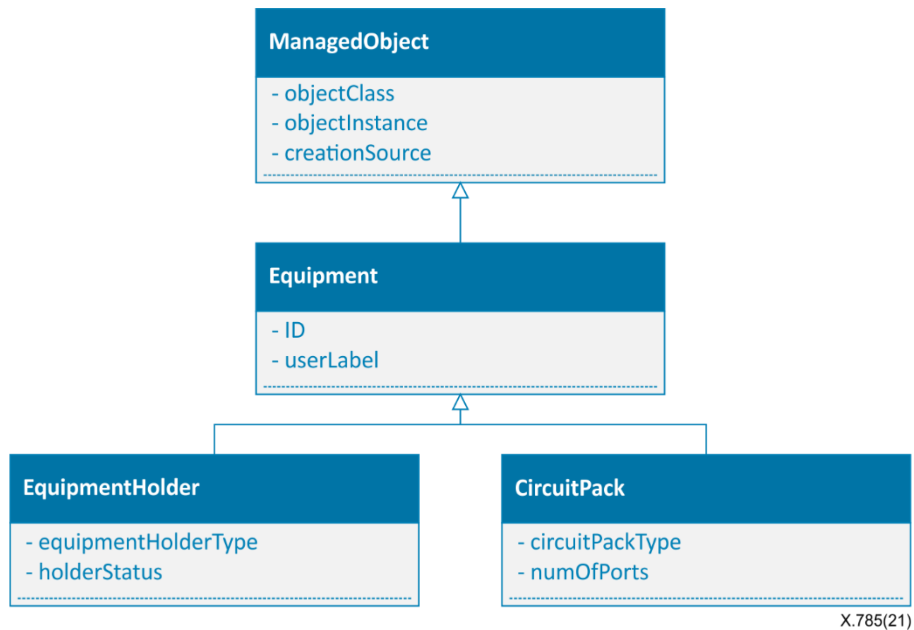

Figure 2 – An example for inheritance relationship

Table 4 – YAML schema example of Equipment and EquipmentHolder by extension

```
Equipment_C:
  allOf:
     - $ref: '#/components/schemas/ManagedObject_C'
     - properties:
         ID:
           type: integer
         userLabel:
           type: string
     - required:
        - ID
        - userLabel
```

```
EquipmentHolder_C:
  allOf:
     - $ref: '#/components/schemas/Equipment_C'
     - properties:
         equipmentHolderType:
           type: string
         holderStatus:
           $ref: '#/components/schemas/HolderStatusType'
     - required:
        - equipmentHolderType
        - holderStatus
```

Some managed objects may need to support multiple inheritance, and the "allOf" syntax of the YAML schema itself supports multiple inheritance. In addition, YAML schema is only used to describe the attribute information of managed objects.

8.2.3 Common attributes and data types

The following table shows some common attributes as well as some common data types that can be shared by this framework.

Table 5 – Standard attributes and data types

Attribute name	Data type	Description
administrativeState	AdministrativeStateType	See [ITU-T M.3701] for more details
availabilityStatus	AvailabilityStatusSetType	See [ITU-T M.3701] for more details
backedUpStatus	BackedUpStatusType	See [ITU-T M.3701] for more details
controlStatus	ControlStatusSetType	See [ITU-T M.3701] for more details
creationSource (Note)	SourceIndicatorType	See [ITU-T M.3701] for more details
externalTime	ExternalTimeType	
objectClass (Note)	String	It indicates an MOC
objectInstance (Note)	URI	It indicates an MO instance
operationalState	OperationalStateType	See [ITU-T M.3701] for more details
proceduralStatus	ProceduralStatusSetType	See [ITU-T M.3701] for more details
standbyStatus	StandbyStatusType	See [ITU-T M.3701] for more details
systemLabel	String	It indicates a label for a system.
unknownStatus	UnknownStatusType	See [ITU-T M.3701] for more details
usageState	UsageState	See [ITU-T M.3701] for more details
NOTE – These attributes are inherited by all managed objects.		

The detailed YAML/JSON definitions for the above data types can be found in Annex A.1.

8.2.4 Containment relationship of managed objects

Different from the inheritance relationship, the containment relationship is more reflected in the affiliation between various network resources. As mentioned above, a switch device may contain several frames, and one frame may contain several racks. A rack may contain a number of slots, and a board that performs various functions is inserted into a slot. There may be various ports on a board. In the containment relationship, an object class (or object instance) used to contain other managed objects is called a superior, and a contained object class (or object instance) is called a subordinate. The names of the superiors and subordinates here are relative, and the subordinates of one object can be the superiors of another object. The relation type ContainmentRelationshipType is defined in YAML schema, which defines six attributes including the relationship name, the superior class name, the superior class multiplicity, the subordinate class name, the subordinate class multiplicity, and the naming attribute, and it is shown in Table 6.

Table 6 – The YAML schema for ContainmentRelationshipType

```
components:
  schemas:
    ContainmentRelationshipType:
      type: object
      properties:
        containmentRelationshipName:
           type: string
        superiorClass:
           type: string
        superiorClassMuitiplicity:
           $ref: '#/components/schemas/MultiplicityType'
        subordinateClass:
           type: string
        subordinateClassMuitiplicity:
```

Table 6 – The YAML schema for ContainmentRelationshipType

```
            $ref: '#/components/schemas/MultiplicityType'
          namingAttrbiute:
            type: string

      DirectionType:
        type: string
        enum:
          - unidirectional
          - bidirectional

      MultiplicityType:
        type: string
        enum:
          - zero_to_one
          - zero_to_n
          - one
          - one_to_n
          - n
```

Bullet (1) in Appendix II shows an example of the usage of ContainmentRelationshipType.

8.2.5 Association relationship of managed objects

In addition to the inheritance and containment relationships, there are also association relationships between managed objects, which are abstractions of network resources. An operation on one managed object may influence the attributes of another one or more managed objects. There are many types of associations, such as business relationships, control relationships, primary and secondary relationships, backup relationships, grouping relationships, peer relationships, and so on. The management system must be able to detect the existence and change of this association, and can align or coordinate the relationship through appropriate operations. Therefore, various association relationships between managed objects must be modelled.

For the definition of association relationship, the YAML schema of AssociationRelationshipType is defined. The attributes of the association relationship include association name, association direction, from association class name, from association attribute name, from association multiplicity, to association class name, to association attribute name, to association multiplicity. The types of the attributes are all string type. The YAML schema definition of the association type is shown in Table 7.

Table 7 – The YAML schema for AssociationRelationshipType

```
components:
  schemas:
    AssociationRelationshipType
      type : object
      properties:
        associationRelationshipName:
           type: string
        associationDirection:
           $ref: '#/components/schemas/DirectionType'
        fromClass:
           type: stirng:
        fromAssociationAttribute
           type: stirng
        fromMuitiplicity:
           $ref: '#/components/schemas/MultiplicityType'
        toClass:
           type: stirng
        toAssociationAttribute:
           type: stirng
        toMuitiplicity:
           type: '#/components/schemas/MultiplicityType'
```

Bullet (2) in Appendix II shows an example about the usage of AssociationRelationshipType.

9 Accessing methods for managed objects

There are two ways of accessing MOs in this framework. The first is to use a generic service that provides the functionality of accessing all kinds of MO instances, where a unique URI of the service is provided, as defined in clause 9.1. The other way is to access specific MOs using their own URIs, and follow the guidelines provide in clause 9.2.

9.1 Generic MO accessing methods

This clause describes the MO accessing methods of the REST-based network management framework, which shall provide a collection of methods to control network resources. These methods provide basic capabilities to manage MOs and so they are called generic accessing methods, as listed in Table 8. Figure 1 gives the accessing procedure and the framework uses HTTP protocol to exchange information of MOs. RESTful services separate an MOs' states and behaviours and expose their behaviours through a RESTful interface. As a RESTful service is also a service-oriented technology, in this framework all MOs are designed to be accessed through the REST interface, and the interface must know which MO is the actual target of an operation, and the unique identifier of the target MO should be provided in each accessing request (or through the URI). According to the above requirements, some necessary generic accessing methods are given in Table 8.

Table 8 – Generic accessing methods

Operation name	Input parameter	Output parameter
getMOAtrributes	– objectClass : String – objectInstance : DN – attributeNameList : SEQUENCE OF String	– attributeNameAndValueList : SEQUENCE OF { attributeName string, attributeType string, attributeValue object} – status : ENUMERATION

Table 8 – Generic accessing methods

Operation name	Input parameter	Output parameter
setMOAttributes	– objectClass : String – objectInstance : DN – attributeNVMList : SEQUENCE OF {attributeName string, attributeType string, attributeValue object}	– status
createMO	– objectclass – objectClassInstance – attributeNameAndValueList	– status
deleteMO	– objectclass – objectInstance	– status

Where:

1) getMOAttributes – to retrieve all, or any subset, of an MO's attribute value in one operation. It uses the URI as the first parameter to uniquely identify the MO and a list of attribute names to be queried. The return result is made up of attribute values and operation status. The attributeNameAndValueList is a list of triples including attributeName, attributeType and attributeValue. The attributeType indicates the original type of attributeValue and attribute values are returned through the "object" element of JSON/YAML schema for arbitrary type values. The status parameter indicates whether the operation is performed successfully or failed. As "object" is defined for the data type of the return attribute value, when receiving such a request from the client, the server will return the requested attributes into the output parameter attributeNameAndValueList, where the attributeValue field will be encoded from a variable element to a piece of JSON text, which can be decoded by the client application with the help of the attributeType parameter.

2) setMOAttributes – to modify attribute values of an MO in existence. Besides using objectInstance to indicate the target MO whose values are to be modified, the operation also uses a list of quadruples including attributeName, attributeType, attributeValue, and modifyOption to set MO attributes. The three attributes are the same as above.

3) createMO – to create an MO in the managed system. It must specify the created MO's class and name. The attributeNameAndValueList parameter is used to provide attribute values, but it can be omitted, and if it is omitted the attributes are set to default values.

4) deleteMO – to release any resources associated with the MO and to delete it. It uses URI to identify the target MO and then return the operation status. If the target MO cannot be removed or any of its contained MOs cannot be removed, the operation will return an OperationFailed status.

All the methods mentioned above just operate on one MO.

A complete YAML/JSON schema and RESTful interface definition for the generic MO accessing methods can be found in clause A.2.

(O) OBJECT-2. An implementation of the generic MO accessing methods may support all the operations described above, whose JSON/YAML schema is defined in clause A.2.

9.2 Design guidelines for specific MO class accessing methods

For each specific kind of MO class, under the REST-based management framework, they should have a unique URI for the instances of the MOC of this kind. The URI for this MOC or instances of this

MOC usually follows the rules specified in Table 3 of clause 8.2.1. Suppose the target MOC is "Equipment" as defined in [ITU-T M.3160], for a collection resource, its URI may look like the following example, as shown in Table 9:

Table 9 – A URI example for a collection resource

```
/CM/cmIpr/v1_0/Network=CoreNetwork/ManagedElement=me1/Equipment
```

For a document resource style MO instance, its URI may look like the following as an example:

Table 10 – A URI example for a document resource

```
/CM/cmIpr/v1_0/Network=CoreNetwork/ManagedElement=me1/Equipment=eq2
```

The above two URIs will be used for the CRUD operations for accessing the specific MOs, and the guidelines will be explained in the following clauses.

9.2.1 Creating a resource instance

When creating a new MO instance of a resource, the HTTP POST method shall be used, and the target URI for this method shall be the URI of the collection resource without any specific identifier, like the example shown in Table 9. The input parameter shall contain all the necessary information for the creation of a resource. On success, the "201 Created" shall be returned, and the complete MO identifier should also be included in the response. On failure, the appropriate error code shall be returned, and the response message body may provide additional error information.

An example of the YAML definition of this operation on a specified resource can be found in Appendix I.1.1.

9.2.2 Reading a group of resource instances by a collection resource

When reading the information of a list of MO instance of a specific kind of resource, the HTTP GET method shall be used, and the target URI for this method shall be the URI of the collection resource to be read without identifiers, like the example shown in Table 9. In such cases, the URI indicates the list of MOs of the same kind of resource, under the same subtree, represented by the parent node of the subtree. There is no need to provide other input parameters in this collection read operation. On success, the "200 OK" shall be returned, and the complete information of all the instances that can be represented by the collection resource shall be included in the response. On failure, the appropriate error code shall be returned, and the response message body may provide additional error information.

An example of the YAML definition of this operation on a collection resource can be found in Appendix I.1.2.

9.2.3 Reading a specific resource instance

When reading the information of a specific MO instance of a resource, the HTTP GET method shall be used, and the target URI for this method shall be the URI of the resource to be read with the specific identifier, like the example shown in Table 10. There is no need to provide other input parameters in this read operation. On success, the "200 OK" shall be returned, and the complete information of the specified MO shall be included in the response. On failure, the appropriate error code shall be returned, and the response message body may provide additional error information.

An example of the YAML definition of this operation on a specified resource can be found in Appendix I.1.3.

9.2.4 Updating a complete representation of a specific resource instance

When updating the information of a complete MO instance of a specific resource, the HTTP PUT method shall be used, and the target URI for this method shall be the URI of the resource to be updated with the specific identifier, like the example shown in Table 10.

The input parameter of this operation shall contain all the attribute information of the specified MO to be updated. On success, the attribute values of the specified target resource shall be replaced by the input parameters (note: partial representations of the resource to be updated are not allowed.), and the "204 No Content" shall be returned. On failure, the appropriate error code shall be returned, and the response message body may provide additional error information. If the specified resource does not exist, the resource code 404 shall be returned.

An example of the YAML definition of this operation on a specified resource can be found in Appendix I.1.4.

9.2.5 Updating partial information of a resource instance

When updating partial information of a MO instance of a specific resource, the HTTP PATCH method shall be used, and the target URI for this method shall be the URI of the resource to be updated with the specific identifier, like the example shown in Table 10.

The input parameters of this operation shall contain the information for partial updates of the resource. The format of a JSON patch or JSON merge patch document describing a set of modification instructions to be applied to the target resource can be found in [RFC 7396]. On success, "200 OK" together with the representation of the updated resource in the message body, or "204 No Content" shall be returned. On failure, the appropriate error code shall be returned, and the response message body may provide additional error information.

An example of the YAML definition of this operation on a specified resource can be found in Appendix I.1.5.

9.2.6 Deleting a resource instance

When deleting a specific resource, the HTTP DELETE method shall be used, and the target URI for this method shall be the URI of the resource to be deleted with the specific identifier, like the example shown in Table 10. There is no need to provide other input parameters in this delete operation. On success, the "204 No Content" shall be returned, and the message body is empty. On failure, the appropriate error code shall be returned, and the response message body may provide additional error information.

An example of the YAML definition of this operation on a specified resource can be found in Appendix I.1.6.

10 Information modelling guidelines for REST-based interfaces

10.1 Resource Modeling

In this Recommendation, MOC represents a resource which is the unit of a collection of attributes. Resource should be defined as a path in YAML/JSON schema.

When using YAML/JSON schema to define the content of an MOC, the YAML *object* is used for modelling the MOC. A YAML/JSON *object* contains a sequence which can include one or more properties. Each *object* corresponding to an MOC should have a "_C" as the suffix to the name.

Other common attribute data types can also be defined as *object,* but they should use the suffix "Type" in the type name.

10.2 Attribute

Attributes and states of an MOC are defined as the *properties* in the *object* corresponding to an MOC.

10.3 Name conventions for MOCs, attributes and data types

The following name conventions are applied for YAML/JSON schema based modelling:

– All the attributes of an MOC are defined as an YAML/JSON object, the name of the MOC should have a "_C" as name suffix, with the first character capitalized, so that this object can be distinguished from other normal data type definitions. For example: ManagedObject_C, Equipment_C.

– An attribute of an MOC is defined as a property within the YAML/JSON object presenting an MOC, and the first character of an attribute name should be in lower case.

– A normal data type definition should have a "Type" as its name suffix, with the first character capitalized to make it more readable. For example: AdminstrativeStateType.

– Use lowerCamelCase, e.g., "personName" for attribute.

– Use UpperCamelCase for defining data type names.

– A set-valued type (unordered set) should have SetType as its name suffix, and a list-valued type (ordered sequence) should have ListType as its name suffix.

10.4 Other guidelines

– Normal operations, including create, retrieve, update and delete (CRUD), should be defined as HTTP operations (post, get, put/patch, and delete).

– Input parameters of a request should be either defined as path parameters, or in the requestBody of the request.

– Possible return values should be defined using different response codes for different branches, which can provide the corresponding meaning descriptions, as well as the return data types.

– Detailed parameters should be defined in the "parameters" part of the JSON/YAML interface file.

– Entity information and concrete data types should be defined in the "definitions" part of the JSON/YAML interface file.

– Tag in an JSON/YAML interface can be used to make several operations for one or more resources as a group, and it will help to organize certain operations for the same purpose.

11 Compliance and conformance

This clause defines the criteria that must be met by other standard documents claiming compliance to these guidelines and the functions that must be implemented by systems claiming conformance to this Recommendation.

11.1 Standards document compliance

Any specification claiming compliance with these guidelines shall:

1) Define all classes that model resources as a derivation (direct or indirect) from the ManagedObject_C described in clause 8.2.1 and defined in the JSON/YAML schema in clause A.1.

2) Support the attributes inheritance using the mechanism specified in clause 8.2.2.

3) Use the definitions for generic attribute types found in clause 8.2.3 wherever applicable.

4) Use the common data types defined in the JSON/YAML schema in clause A.1 whenever appropriate.

5) Define specific management information models and REST-based interface following the guidelines and conventions specified in clause 10.

11.2 System conformance

An implementation claiming conformance to this Recommendation shall:

1) Either support all of the capabilities of the generic MO accessing methods as described in clause 9.1, and support the corresponding REST interface as defined in clause A.2; or

2) Support specific MO accessing methods following the guidelines as defined in clause 9.2.

11.3 Conformance statement guidelines

The conformance statement must identify a document and year of publication to make sure the right version of YAML/JSON schema is identified.

Annex A

Common REST-based YAML/JSON schema definitions

(This annex forms an integral part of this Recommendation.)

In this Annex, the common definitions of REST interfaces as well as some common JSON/YAML schema based data types are defined.

A.1 YAML schema definitions for the generic managed object and common data types

Table A.1 provides the YAML schema of the generic MOC ManagedObject_C, and the common attribute data types that can be shared when defining specific interface information models.

Table A.1 – YAML schema for ManagedObject_C and common data types


```
components:
  schemas:
    ManagedObject_C:
      type: object
      required:
      - objectClass
      - objectInstance
      properties:
        objectClass:
          type: string
        objectInstance:
          type: string
          format: uri
        creationSource:
          $ref: '#/components/schemas/SourceIndicatorType'

    SourceIndicatorType:
      type: string
      enum:
        - resourceOperation
        - managementOperation
        - unknown

    ContainmentRelationshipType:
      type: object
      properties:
        associationRelationshipName:
           type: string
        associationDirection:
           $ref: '#/components/schemas/DirectionType'
        fromClass:
           type: string
        fromAssociationAttribute:
           type: string
        fromMuitiplicity:
           $ref: '#/components/schemas/MultiplicityType'
        toClass:
           type: string
        toAssociationAttribute:
           type: string
        toMuitiplicity:
           $ref: '#/components/schemas/MultiplicityType'
```

```
    AssociationRelationshipType:
      type: object
      properties:
        associationRelationshipName:
           type: string
        associationDirection:
           $ref: '#/components/schemas/DirectionType'
        fromClass:
           type: string
        fromAssociationAttribute:
           type: string
        fromMuitiplicity:
           $ref: '#/components/schemas/MultiplicityType'
        toClass:
           type: string
        toAssociationAttribute:
           type: string
        toMuitiplicity:
           $ref: '#/components/schemas/MultiplicityType'

    MultiplicityType:
      type: string
      enum:
        - zero_to_one
        - zero_to_n
        - one
        - one_to_n
        - n

    DirectionType:
      type: string
      enum:
        - unidirectional
        - bidirectional

    AdministrativeStateType:
      type: string
      enum:
        - locked
        - unlocked
        - shuttingDown

    AvailabilityStatusType:
      type: string
      enum:
        - inTest
        - failed
        - powerOff
        - offLine
        - offDuty
        - dependency
        - degraded
        - notInstalled
        - logFull

    AvailabilityStatusSetType:
      type: array
      items:
        $ref: '#/components/schemas/AvailabilityStatusType'
```

```
    BackedUpStatusType:
      type: boolean

    ControlStatusType:
      type: string
      enum:
        - inTestsubjectToTest
        - partOfServicesLocked
        - reservedForTest
        - suspended

    ControlStatusSetType:
      type: array
      items:
        $ref: '#/components/schemas/ControlStatusType'

    ExternalTimeType:
      type: string
      format: dateTime

    OperationalStateType:
      type: string
      enum:
        - disabled
        - enabled

    ProceduralStatusType:
      type: string
      enum:
        - initializationRequired
        - notInitialized
        - initializing
        - reporting
        - terminating

    ProceduralStatusSetType:
      type: array
      items:
        $ref: '#/components/schemas/ProceduralStatusType'

    StandbyStatusType:
      type: string
      enum:
        - hotStandby
        - coldStandby
        - providingService

    UnknownStatusType:
      type: boolean

    UsageStateType:
      type: string
      enum:
        - idle
        - active
        - busy
```

A.2 YAML/JSON schema definitions for common object accessing methods

This clause provides the generic MO accessing methods and also provides the JSON schema definitions used in these operations.

[RFC 7231], [RFC 7232] and [RFC 6585] provide dozens of definitions for status codes, which will likely be referenced in implementations of this Recommendation. In summary, these status codes can be found in Table A.2.

Table A.2 – Definitions of status code

Category	Status code	Definition	Reference
Informational 1xx	100	Continue	[RFC 7231]
	101	Switching protocols	[RFC 7231]
Successful 2xx	200	OK	[RFC 7231]
	201	Created	[RFC 7231]
	202	Accepted	[RFC 7231]
	203	Non-authoritative information	[RFC 7231]
	204	No content	[RFC 7231]
	205	Reset content	[RFC 7231]
Redirection 3xx	300	Multiple choices	[RFC 7231]
	301	Moved permanently	[RFC 7231]
	302	Found	[RFC 7231]
	303	See other	[RFC 7231]
	304	Not modified	[RFC 7232]
	305	Use proxy	[RFC 7231]
	306	(Unused)	[RFC 7231]
	307	Temporary redirect	[RFC 7231]
Client error 4xx	400	Bad request	[RFC 7231]
	402	Payment required	[RFC 7231]
	403	Forbidden	[RFC 7231]
	404	Not found	[RFC 7231]
	405	Method not allowed	[RFC 7231]
	406	Not acceptable	[RFC 7231]
	408	Request timeout	[RFC 7231]
	409	Conflict	[RFC 7231]
	410	Gone	[RFC 7231]
	411	Length required	[RFC 7231]
	412	Precondition failed	[RFC 7232]
	413	Request entity too large	[RFC 7231]
	414	Request-URI too long	[RFC 7231]
	415	Unsupported media type	[RFC 7231]
	417	Expectation failed	[RFC 7231]
	426	Upgrade required	[RFC 7231]
	428	Precondition required	[RFC 6585]

Table A.2 – Definitions of status code

Category	Status code	Definition	Reference
	429	Too many requests	[RFC 6585]
	431	Request header fields too large	[RFC 6585]
Server error 5xx	500	Internal server error	[RFC 7231]
	501	Not implemented	[RFC 7231]
	502	Bad gateway	[RFC 7231]
	503	Service unavailable	[RFC 7231]
	504	Gateway timeout	[RFC 7231]
	505	HTTP version not supported	[RFC 7231]
	511	Network authentication required	[RFC 6585]

A.2.1 createMO operation

The createMO operation is used to create an instance of an MO class. The REST interface definition for this operation contains the following:

(1) The request URL: the URI of the generic MOAccessService. This operation is equivalent to adding a new instance of a managed resource in a collection resource.

(2) The HTTP method: the create resource operation should be mapped to the "POST" request method.

(3) The content of the request body: the createMO operation needs to give all the attribute names and the corresponding values of the managed object instance (MOI) to be created in the request body. The request body type is CreateMORequest, and its JSON Schema type inherits from the ManagedObject_C base class.

(4) Possible response codes and corresponding response body contents: the possible status codes for the operation are 201, 400, 404, 405, 409, and 500. When the status code is 201, the creation is successful, and the MO ID will be returned in the response body. The URI identifier of the newly created object instance needs to be given in the location header field of the response, and all attribute information of the newly created object is returned in the response body. In general, the status code 404 indicates the URL does not exist; the status code 405 indicates this operation is not allowed for the specified resource; the status code 500 indicates there is an internal server error. The status code 409 indicates that there is a conflict. If the ID of the newly created managed object is determined by the client side, there may be a 409 conflict; if it is determined by the server side, there will be no 409 conflict. When the response status code is 400, the error information returned in the response body is CreateMOErrorInfo, and the enumeration values of the error type in the code attribute information include: objectClassSpecificationMissmatched, InvalidObjectInstance, noSuchObjectClass, noSuchAttribute, invalidAttributeValue, missingAttributeValue, and the JSON schema is of the CreateMOErrorInfo type. Based on the above analysis, the JSON schema interface definition of the createMO operation can be found in Table A.3.

Table A.3 – JSON schema definition of createMO operation

Name	JSON schema
REQUEST	`"POST" "/MOAccessService" {HTTPVersion} {RequestBody}`
RequestBody	`CreateMORequest : extends ManagedObject_C`

RESPONSE	`{HTTPVersion} {StatusCode} {ReasonPhrase}`
	`Location: {URI}`
	`{ResponseBody}`

The YAML schema definitions of the createMO operation can be defined in Table A.4.

Table A.4 – YAML schema definition of createMO operation

```
paths:
  /MOAccessService:
    post:
      tags:
      - MOAccessService
      summary: "createMO"
      description: "create an MO instance with the specified attribute list"
      operationId: "createMO"
      requestBody:
        description: "The input parameters of the createMO operation"
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/CreateMORequest'
        required: true
      responses:
        201:
          description: "MO is successfully created, and the new MO ID value 
will be returned."
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/MOID'
        400:
          description: "Parameter Error occurred in the createMO operation"
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/CreateMOErrorInfo'
        404:
          description: "The URL does not exist"
          content: {}
        405:
          description: "Method Not Allowed"
          content: {}
        409:
          description: "Conflict MO Id"
          content: {}
        500:
          description: "Internal Server Error"
          content: {}

components:
  schemas:
```

```
    MOInfo:
      type: object
      properties:
        moInfo:
          $ref: '#/components/schemas/ManagedObject_C'
        attributeList:
          type: array
          items:
            $ref: '#/components/schemas/NVPair'

    NVPair:
      type: object
      required:
      - name
      - value
      properties:
        name:
          type: string
        value:
          type: string
        type:
          type: string

    NVPairList:
      type: object
      properties:
        attributeList:
          type: array
          items:
            $ref: '#/components/schemas/NVPair'

    CreateMORequest:
      allOf:
        - $ref: '#/components/schemas/ManagedObject_C'
        - $ref: '#/components/schemas/NVPairList'

    CreateMOErrorInfo:
      type: object
      properties:
        code:
          type: string
          enum:
            - objectClassSpecificationMissmatched
            - invalidObjectInstance
            - noSuchObjectClass
            - noSuchAttribute
            - invalidAttributeValue
            - missingAttributeValue
        message:
          type: string
      required:
        - code

    MOID:
      type: string
      format: uri
```

A.2.2 getMOAttributes operation

The getMOAttributes operation is used to get all or part of the attribute values of an MO instance. The REST interface definition for this operation contains the following:

(1) The request URL: the URI of the generic MOAccessService. This operation supports querying all attribute values or partial attribute values of a managed object instance. The input parameters contain object class, and MO instance, which identifies the managed object to be queried. If the client needs to query partial attribute values, the list of attribute names to be queried should be listed in the query parameters in the request body; otherwise the response returns all the attribute values of the specified MO;

(2) The HTTP method: this operation is a query operation and should be mapped to the 'GET' request method.

(3) Possible response codes and corresponding response body contents: the possible status codes of the method are 200, 400, 404, and 500. When the request is successful, the response status code is 200, and the object containing the attribute name and value pairs is returned in the response body, and the JSON schema is of the "MOInfo" type. When the response status code is 400, the response body will give the error reason for the request failure. The error information type GetMOAttributesErrorInfo of the method gives the code attribute information unique to the method. The enumerated values include four types: duplicateInvocation, resourceLimitation, operationCancelled, and complexityLimitation. The status code 404 indicates the MO instance does not exist; the status code 500 indicates there is an internal server error.

Based on the above analysis, the JSON schema interface definition of the getMOAttributes operation can be found in Table A.5.

Table A.5 – JSON schema definition of getMOAttributes operation

Name	JSON schema
REQUEST	"GET" "/MOAccessService" {HTTPVersion} {Parameters}
Parameters	{objectClass, moInstance, [{attributeNameList}]}
attributeNameList	{attributeName} ("," {attributeName})*
RESPONSE	{HTTPVersion} {StatusCode} {ReasonPhrase} {ResponseBody}
ResponseBody	MOInfo: object

The YAML schema definitions of the getMOAttributes operation can be defined in Table A.6.

Table A.6 – YAML schema definition of getMOAttributes operation

```
paths:
  /MOAccessService:
    get:
      tags:
      - MOAccessService
      summary: "getMOAttributes"
      description: "get the attributes information of a specific MO instance"
      operationId: "getMOAttributes"
      parameters:
      - name: objectClass
        in: query
        description: "The Obejct class of the specified MO instance"
        required: true
        schema:
          type: string
      - name: moInstance
        in: query
        description: "The unique ID of the specific MO instance"
        required: true
```

```
        schema:
          type: string
          format: uri
      - name: attributeNameList
        in: query
        description: "the attribute name of the string"
        required: false
        schema:
          type: array
          items:
            $ref: '#/components/schemas/attributeName'
      responses:
        200:
          description: "The attribute value list of the specified MO is retrieved
successfully"
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/MOInfo"
        400:
          description:  "Parameter  Error  occured  in  the  getMOAttributes
operation"
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/GetMOErrorInfo"
        404:
          description: "Specified MO Not found"
          content: {}
        500:
          description: "Internal Server Error"
          content: {}

components:
  schemas:
    GetMOErrorInfo:
      type: object
      properties:
        code:
          type: string
          enum:
            - duplicateInvocation
            - resourceLimitation
            - operationCancelled
            - complexityLimitation
        message:
          type: string
      required:
        - code

    attributeName:
      type: string
```

A.2.3 setMOAttributes operation

The setMOAttributes operation is used to set or modify all or part of the attribute values of an MO instance. The REST interface definition for this operation contains the following:

(1) The HTTP method: this is an operation to update the resource attribute, and supports partial updating of the MO attributes, so it should be mapped to the 'PATCH' request method.

(2) The content of the request body: the setMOAttributes operation needs to give the attribute name and value pairs of the MO instance to be modified in the request body, and the JSON schema of the request body is of an "object" type.

(3) Possible response codes and corresponding response body contents: the possible status codes of the method are 200, 204, 400, 404, 405, and 500. The 200 status code indicates that the operation has been successfully processed, maybe with some minor changes, and the response body contains the updated new values of the specified MO instance; the 204 status code indicates that the operation has been successfully processed without any changes, and no response is needed. When the status code is 400, the error information returned in the response body is SetMOAttributesErrorInfo, and the enumeration values of the error type in the code attribute information include modifyNotAllowed, noSuchAttribute, invalidAttributeValue, missingAttributeValue, complexityLimitation. The status code 404 indicates the MO instance does not exist; the status code 405 indicates this operation is not allowed for the specified resource; the status code 500 indicates there is an internal server error.

Based on the above analysis, the JSON schema definition of the setMOAttributes operation can be found in Table A.7.

Table A.7 – JSON schema definition of setMOAttributes operation

Name	JSON schema
REQUEST	`"PATCH" {"/MOAccessService"} {HTTPVersion}{RequestBody}`
RequestBody	`MOInfo: object`
RESPONSE	`{HTTPVersion} {StatusCode} {ReasonPhrase}`

The YAML schema definitions of the setMOAttributes operation is defined in Table A.8.

Table A.8 – YAML schema definition of setMOAttributes operation

```
paths:
  /MOAccessServie
    patch:
      tags:
      - MOAccessService
      summary: Set attribute values of an MO instance
      operationId: setMOAttributes
      requestBody:
        description: MO information that is used for modification
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/MOInfo'
        required: true
      responses:
        200:
          description: "The specified MO instance attributes are updated
successfullly."
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/MOInfo'
        204:
          description: "MO successfully modified, without any change. No
```

```
response is needed."
        400:
          description: "Parameter Error occurred in the setMOAttributes
operation"
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/SetMOAttributesErrorInfo"
        404:
          description: Specific MO does not exist.
          content: {}
        405:
          description: Validation parameter
          content: {}
        500:
          description: "Internal Server Error"
          content: {}

components:
  schemas:
    SetMOAttributesErrorInfo:
      type: object
      properties:
        code:
          type: string
          enum:
            - modifyNotAllowed
            - noSuchAttribute
            - invalidAttributeValue
            - missingAttributeValue
            - complexityLimitation
        message:
          type: string
      required:
        - code
```

A.2.4 deleteMO operation

The deleteMO operation is used to delete an MO instance. The REST interface definition for this operation contains the following:

(1) The request URL: the URI of the generic MOAccessService. This operation supports deleting a specified managed object instance. The input parameters contain the object class, and MO instance to be deleted.

(2) The HTTP method: this operation is to delete a resource operation, so it should be mapped to the 'DELETE' request method.

(3) Possible response codes and corresponding response body contents: The possible status codes of the method are 200, 400, 404, 405, and 500. When the response status code is 400, the error information returned in the response body is DeleteMOErrorInfo, and the enumeration values of the error type in the code attribute information include resourceLimitation and complexityLimitation. The status code 404 indicates the MO instance does not exist; the status code 405 indicates the "DELETE" operation is not allowed for the specified resource; the status code 500 indicates there is an internal server error. Based on the above analysis, the JSON schema definition of the deleteMO operation can be found in Table A.9.

Table A.9 – JSON schema definition of deleteMO operation

Name	JSON schema
REQUEST	`"DELETE" "/MOAccessService" {HTTPVersion} {RequestBody}`
RequestBody	`DeleteMORequest : object`
RESPONSE	`{HTTPVersion} {StatusCode} {ReasonPhrase}`

The YAML schema definitions of the deleteMO operation can be defined in Table A.10.

Table A.10 – YAML schema definition of deleteMO operation

```
paths:
  /MOAccessService:
    delete:
      tags:
      - MOAccessService
      summary: "deleteMO"
      description: "delete a specific MO instance"
      operationId: "deleteMO"
      parameters:
      - name: objectClass
        in: query
        description: "The Object class of the specified MO instance"
        required: true
        schema:
          type: string
      - name: moInstance
        in: query
        description: "The unique ID of the specific MO instance"
        required: true
        schema:
          type: string
          format: uri
      responses:
        200:
          description: "The specified MO instance is deleted successfullly"
          content: {}
        400:
          description: "Parameter Error occurred in the deleteMO operation"
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/DeleteMOErrorInfo'
        404:
          description: "Specified MO does not exist"
          content: {}
        405:
          description: "Method Not Allowed, the specified MO cannot be deleted"
          content: {}
        500:
          description: "Internal Server Error"
          content: {}

components:
  schemas:
    DeleteMOErrorInfo:
      type: object
      properties:
        code:
```

```
          type: string
          enum:
            - resourceLimitation
            - complexityLimitation
        message:
          type: string
      required:
        - code
```

A.2.5 Complete definitions of the MO access service

In summary, the REST interface definition of the four operations of the MO access service can be found in Table A.11.

Table A.11 – Complete interface definition of MOAccessService operations

```
openapi: 3.0.0
info:
  title: ITU-T_MOAccessService
  description: 'This is the formal definition of the generic MO Access Service.'
  termsOfService: https://www.itu.int/en/ITU-T/about/Pages/default.aspx
  contact:
    email: zlwang@bupt.edu.cn
  license:
    name: Apache 2.0
    url: http://www.apache.org/licenses/LICENSE-2.0.html
  version: 1.0.0
externalDocs:
  description: See Recommendation ITU-T X.785
  url: https://www.itu.int/itu-t/recommendations/index.aspx?ser=X
servers:
- url: https://www.itu.int/demo/genericMOAccessService/v1.0.0
- url: http://www.itu.int/demo/genericMOAccessService/v1.0.0
tags:
- name: MOAccessService
  description: All APIs realted to the generic MO Access Service
  externalDocs:
    description: Find out more
    url: https://www.itu.int/itu-t/recommendations/index.aspx?ser=X

paths:
  /MOAccessService:
    post:
      tags:
      - MOAccessService
      summary: "createMO"
      description: "create an MO instance with the specified attribute list"
      operationId: "createMO"
      requestBody:
        description: "The input parameters of the createMO operation"
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/CreateMORequest'
        required: true
      responses:
        201:
          description: "MO is successfully created, and the new MO ID value will
be returned."
          content:
```

```
            application/json:
              schema:
                $ref: '#/components/schemas/MOID'
        400:
          description: "Parameter Error occurred in the createMO operation"
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/CreateMOErrorInfo'
        404:
          description: "The URL does not exist"
          content: {}
        405:
          description: "Method Not Allowed"
          content: {}
        409:
          description: "Conflict MO Id"
          content: {}
        500:
          description: "Internal Server Error"
          content: {}

    get:
      tags:
      - MOAccessService
      summary: "getMOAttributes"
      description: "get the attributes information of a specific MO instance"
      operationId: "getMOAttributes"
      parameters:
      - name: objectClass
        in: query
        description: "The Obejct class of the specified MO instance"
        required: true
        schema:
          type: string
      - name: moInstance
        in: query
        description: "The unique ID of the specific MO instance"
        required: true
        schema:
          type: string
          format: uri
      - name: attributeNameList
        in: query
        description: "the attribute name of the string"
        required: false
        schema:
          type: array
          items:
            $ref: '#/components/schemas/attributeName'
      responses:
        200:
          description: "The attribute value list of the specified MO is retrieved
successfully"
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/MOInfo"
        400:
```

```
          description:  "Parameter  Error  occured  in  the  getMOAttributes
operation"
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/GetMOErrorInfo"
        404:
          description: "Specified MO Not found"
          content: {}
        500:
          description: "Internal Server Error"
          content: {}

    delete:
      tags:
      - MOAccessService
      summary: "deleteMO"
      description: "delete a specific MO instance"
      operationId: "deleteMO"
      parameters:
      - name: objectClass
        in: query
        description: "The Object class of the specified MO instance"
        required: true
        schema:
          type: string
      - name: moInstance
        in: query
        description: "The unique ID of the specific MO instance"
        required: true
        schema:
          type: string
          format: uri
      responses:
        200:
          description: "The specified MO instance is deleted successfullly"
          content: {}
        400:
          description: "Parameter Error occurred in the deleteMO operation"
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/DeleteMOErrorInfo'
        404:
          description: "Specified MO does not exist"
          content: {}
        405:
          description: "Method Not Allowed, the specified MO cannot be deleted"
          content: {}
        500:
          description: "Internal Server Error"
          content: {}

    patch:
      tags:
      - MOAccessService
      summary: Set attribute values of an MO instance
      operationId: setMOAttributes
      requestBody:
        description: MO information that is used for modification
```

```
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/MOInfo'
        required: true
      responses:
        200:
          description: "The specified MO instance attributes are updated
successfullly."
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/MOInfo'
        204:
          description: "MO successfully modified, without any change. No
response is needed."
        400:
          description: "Parameter Error occurred in the setMOAttributes
operation"
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/SetMOAttributesErrorInfo"
        404:
          description: Specific MO does not exist.
          content: {}
        405:
          description: Validation parameter
          content: {}
        500:
          description: "Internal Server Error"
          content: {}

components:
  schemas:
    ManagedObject_C:
      type: object
      required:
      - objectClass
      - objectInstance
      properties:
        objectClass:
          type: string
        objectInstance:
          type: string
          format: uri
        creationSource:
          $ref: '#/components/schemas/SourceIndicatorType'

    SourceIndicatorType:
      type: string
      enum:
        - resourceOperation
        - managementOperation
        - unknown

    ContainmentRelationshipType:
      type: object
      properties:
        associationRelationshipName:
```

```
        type: string
      associationDirection:
        $ref: '#/components/schemas/DirectionType'
      fromClass:
        type: string
      fromAssociationAttribute:
        type: string
      fromMuitiplicity:
        $ref: '#/components/schemas/MultiplicityType'
      toClass:
        type: string
      toAssociationAttribute:
        type: string
      toMuitiplicity:
        $ref: '#/components/schemas/MultiplicityType'

  AssociationRelationshipType:
    type: object
    properties:
      associationRelationshipName:
        type: string
      associationDirection:
        $ref: '#/components/schemas/DirectionType'
      fromClass:
        type: string
      fromAssociationAttribute:
        type: string
      fromMuitiplicity:
        $ref: '#/components/schemas/MultiplicityType'
      toClass:
        type: string
      toAssociationAttribute:
        type: string
      toMuitiplicity:
        $ref: '#/components/schemas/MultiplicityType'

  MultiplicityType:
    type: string
    enum:
      - zero_to_one
      - zero_to_n
      - one
      - one_to_n
      - n

  DirectionType:
    type: string
    enum:
      - unidirectional
      - bidirectional

  AdministrativeStateType:
    type: string
    enum:
      - locked
      - unlocked
      - shuttingDown

  AvailabilityStatusType:
    type: string
```

```
      enum:
        - inTest
        - failed
        - powerOff
        - offLine
        - offDuty
        - dependency
        - degraded
        - notInstalled
        - logFull

    AvailabilityStatusSetType:
      type: array
      items:
        $ref: '#/components/schemas/AvailabilityStatusType'

    BackedUpStatusType:
      type: boolean

    ControlStatusType:
      type: string
      enum:
        - inTestsubjectToTest
        - partOfServicesLocked
        - reservedForTest
        - suspended

    ControlStatusSetType:
      type: array
      items:
        $ref: '#/components/schemas/ControlStatusType'

    ExternalTimeType:
      type: string
      format: dateTime

    OperationalStateType:
      type: string
      enum:
        - disabled
        - enabled

    ProceduralStatusType:
      type: string
      enum:
        - initializationRequired
        - notInitialized
        - initializing
        - reporting
        - terminating

    ProceduralStatusSetType:
      type: array
      items:
        $ref: '#/components/schemas/ProceduralStatusType'

    StandbyStatusType:
      type: string
      enum:
        - hotStandby
```

```
      - coldStandby
      - providingService

  UnknownStatusType:
    type: boolean

  UsageStateType:
    type: string
    enum:
      - idle
      - active
      - busy

  MOInfo:
    type: object
    properties:
      moInfo:
        $ref: '#/components/schemas/ManagedObject_C'
      attributeList:
        type: array
        items:
          $ref: '#/components/schemas/NVPair'

  NVPair:
    type: object
    required:
    - name
    - value
    properties:
      name:
        type: string
      value:
        type: string
      type:
        type: string

  NVPairList:
    type: object
    properties:
      attributeList:
        type: array
        items:
          $ref: '#/components/schemas/NVPair'

  CreateMORequest:
    allOf:
      - $ref: '#/components/schemas/ManagedObject_C'
      - $ref: '#/components/schemas/NVPairList'

  CreateMOErrorInfo:
    type: object
    properties:
      code:
        type: string
        enum:
          - objectClassSpecificationMissmatched
          - invalidObjectInstance
          - noSuchObjectClass
          - noSuchAttribute
          - invalidAttributeValue
```

```yaml
          - missingAttributeValue
      message:
        type: string
    required:
      - code

  GetMOErrorInfo:
    type: object
    properties:
      code:
        type: string
        enum:
          - duplicateInvocation
          - resourceLimitation
          - operationCancelled
          - complexityLimitation
      message:
        type: string
    required:
      - code

  SetMOAttributesErrorInfo:
    type: object
    properties:
      code:
        type: string
        enum:
          - modifyNotAllowed
          - noSuchAttribute
          - invalidAttributeValue
          - missingAttributeValue
          - complexityLimitation
      message:
        type: string
    required:
      - code

  DeleteMOErrorInfo:
    type: object
    properties:
      code:
        type: string
        enum:
          - resourceLimitation
          - complexityLimitation
      message:
        type: string
    required:
      - code

  attributeName:
    type: string

  MOID:
    type: string
    format: uri
```

Appendix I

An example of REST-based interface definitions for resource

(This appendix does not form an integral part of this Recommendation.)

I.1 An example showing the CRUD definitions for a specific resource

An example of the YAML definitions of CRUD operations for a specific resource indicating a kind of MO is illustrated in this appendix.

Suppose the target MOC is equipment as described in [ITU-T M.3160], in order to show the entity information as an example, a simplified attributes definition may contain the following list, as shown in Table I.1.

Table I.1 – Attribute list of MOC equipment

Attribute name	Support qualifier	Type
equipmentId	M	String
serialNumber	M	String
locationName	O	String
userLabel	O	String
vendorName	O	String

Part of the YAML entity for Equipment_C and PartialEquipmentAttributeType definitions may look like the following:

```
components:
  schemas:
    Equipment_C:
      allOf:
        - $ref: '#/components/schemas/ManagedObject_C'
        - properties:
            equipmentId:
              type: string
            serialNumber:
              type: string
            locationName:
              type: string
            userLabel:
              type: string
            vendorName:
              type: string
        - required:
          - equipmentId
          - serialNumber

    PartialEquipmentAttributeType:
      type: object
      properties:
        serialNumber:
          type: string
        locationName:
          type: string
        userLabel:
          type: string
        vendorName:
          type: string
```

According to clause 9.2, the collection resource URI example of "Equipment" is shown in Table 9; and the document resource URI example of "Equipment" is shown in Table 10. There are several operations for the access of equipment instances that will be shown in the following sections.

I.1.1 Creating an "Equipment" resource instance

Corresponding to clause 9.2.1, the following example for creating an "Equipment" resource instance can be shown as the following:

```
paths:
  /Network={networkId}/ManagedElement={manageElementId}/Equipment:
    post:
      tags:
      - Equipment
      summary: Create an instance of Equipment
      operationId: createEquipment
      parameters:
      - name: networkId
        in: path
        description: ID of Network
        required: true
        schema:
          type: string
      - name: manageElementId
        in: path
        description: ID of ManagedElement
        required: true
        schema:
          type: string
      requestBody:
        description: Equipment MO information that needs to be created
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Equipment_C'
        required: true
      responses:
        201:
          description: Successfully Created
          content: {}
        400:
          description: Invalid ID supplied
          content: {}
        405:
          description: Validation exception
          content: {}
```

I.1.2 Reading a group of "Equipment" resource instances by a collection resource

Corresponding to clause 9.2.2, the following example for reading a group of "Equipment" resource instances by a collection resource can be shown as the following:

```
paths:
  /Network={networkId}/ManagedElement={manageElementId}/Equipment:
    get:
      tags:
      - Equipment
      summary: Get a list of Equipment under the specific parent MO
      description: On success, the list of Equipment entity information will be
returned.
      operationId: getEquipmentList
```

```
      parameters:
      - name: networkId
        in: path
        description: ID of Network
        required: true
        schema:
          type: string
      - name: manageElementId
        in: path
        description: ID of ManagedElement
        required: true
        schema:
          type: string
      responses:
        200:
          description: successful operation
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Equipment_C'
        404:
          description: Specific parent MO does not exist.
          content: {}
```

I.1.3 Reading a specific "Equipment" resource instance

Corresponding to clause 9.2.3, the following example for reading a specific "Equipment" resource instances can be shown as the following:

```
paths:

/Network={networkId}/ManagedElement={manageElementId}/Equipment={equipmentId}:
    get:
      tags:
      - Equipment
      summary: Get an Equipment instance under the specific parent MO
      description: On success, the specified Equipment entity information will
be returned.
      operationId: getEquipment
      parameters:
      - name: networkId
        in: path
        description: ID of Network
        required: true
        schema:
          type: string
      - name: manageElementId
        in: path
        description: ID of ManagedElement
        required: true
        schema:
          type: string
      - name: equipmentId
        in: path
        description: ID of Equipment
        required: true
        schema:
          type: string
      responses:
        200:
```

```
          description: successful operation
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Equipment_C'
        404:
          description: Specific Equipment MOI does not exist.
          content: {}
```

I.1.4 Updating a complete representation of a specific "Equipment" resource instance

Corresponding to clause 9.2.4, the following example for updating a complete representation of a specific "Equipment" resource instance can be shown as the following:

```
paths:

/Network={networkId}/ManagedElement={manageElementId}/Equipment={equipmentId}:
    put:
      tags:
      - Equipment
      summary: Replace an instance of Equipment
      operationId: replaceEquipment
      parameters:
      - name: networkId
        in: path
        description: ID of Network
        required: true
        schema:
          type: string
      - name: manageElementId
        in: path
        description: ID of ManagedElement
        required: true
        schema:
          type: string
      - name: equipmentId
        in: path
        description: ID of Equipment
        required: true
        schema:
          type: string
      requestBody:
        description: Equipment MO information that is used for replacement
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Equipment_C'
        required: true
      responses:
        200:
          description: MO successfully replaced, there may include some changes,
and the new MO value will be returned.
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Equipment_C'
        204:
          description: MO successfully replaced, without any change.

        404:
          description: Specific MO does not exist.
          content: {}
        405:
```

```
          description: Validation parameter
          content: {}
```

I.1.5 Updating partial information of a specific "Equipment" resource

Corresponding to clause 9.2.5, the following example for updating partial information of a specific "Equipment" resource instance can be shown as the following:

```
paths:

/Network={networkId}/ManagedElement={manageElementId}/Equipment={equipmentId}:
    patch:
      tags:
      - Equipment
      summary: Set partial attribute value of an Equipment instance
      operationId: setEquipment
      parameters:
      - name: networkId
        in: path
        description: ID of Network
        required: true
        schema:
          type: string
      - name: manageElementId
        in: path
        description: ID of ManagedElement
        required: true
        schema:
          type: string
      - name: equipmentId
        in: path
        description: ID of Equipment
        required: true
        schema:
          type: string
      requestBody:
        description: Equipment MO information that is used for replacement
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/PartialEquipmentAttributeType'
        required: true
      responses:
        200:
          description: MO attribute values successfully modified, and the new
MO value will be returned.
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Equipment_C'
        204:
          description: MO successfully modified, without any change. No response
is needed.

        404:
          description: Specific MO does not exist.
          content: {}
        405:
          description: Validation parameter
          content: {}
```

I.1.6 Deleting an "Equipment" resource

Corresponding to clause 9.2.6, the following example for deleting a specific "Equipment" resource instance can be shown as the following:

```
paths:
 /Network={networkId}/ManagedElement={manageElementId}/Equipment=
{equipmentId}:
    delete:
      tags:
      - Equipment
      summary: Deletes an instance of Equipment
      operationId: deleteEquipment
      parameters:
      - name: networkId
        in: path
        description: ID of Network
        required: true
        schema:
          type: string
      - name: manageElementId
        in: path
        description: ID of ManagedElement
        required: true
        schema:
          type: string
      - name: equipmentId
        in: path
        description: ID of Equipment
        required: true
        schema:
          type: string
      responses:
        204:
          description: successful operation
        404:
          description: Specific Equipment does not exist.
          content: {}
        405:
          description: Specific Equipment cannot be deleted.
          content: {}
```

Following the above examples, an interface specification author can develop application-specific REST-based interface operations for accessing specific resources.

Appendix II

Usage examples of the ContainmentRelationshipType and AssociationRelationType

(This appendix does not form an integral part of this Recommendation.)

In clauses 8.2.4 and 8.2.5, the definition of containment and association relationship of managed objects are provided, and the ContainmentRelationshipType is defined. This appendix will show an example on the usage of such types.

The ContainmentRelationshipType is used to express the containment relationship in the resource model. Traditionally in guidelines for the definition of managed objects (GDMO), the containment information is expressed in syntax by Name-Binding, but in REST, there is no syntax to express this information. Usually, there are also UML diagrams associated with the information model, the containment and association relationship can be expressed in UML, but if only data types are left in REST interface definitions, the relationship information might be lost. In such cases, the "ContainmentRelationshipType" and "AssociationRelationshipType" can be used in a formal way for resource information model so that a program can parse such information immediately. The information expressed using ContainmentRelationshipType and the AssociationRelationType is equivalent to the information expressed in UML entity-relationship diagrams.

(1) An example for the usage of ContainmentRelationshipType is given in clause 8.2.4

For example, suppose the MOC EquipmentHolder may contain MOC CircuitPack (See [ITU-T M.3160]), then a JSON instance of ContainmentRelationshipType may be provided as shown in Table II.1.

Table II.1 – An example of ContainmentRelationshipType

```
{
     “containmentRelationshipName”: “EquipmentHolder-CircuitPack-Containment”
     “superiorClass: “EquipmentHolder”
     “superiorClassMuitiplicity”: “one”
     “subordinateClass”: “CircuitPack”
     “subordinateClassMuitiplicity”: “zero_to_n”
     “namingAttrbiute”: “circuitPackId”
}
```

The above example can be explained like this: one instance of MOC EquipmentHolder may contain zero or multiple instances of MOC CircuitPack, and the naming attribute of CircuitPack is circuitPackId. This is equivalent to the UML diagram as shown in Figure II.1.

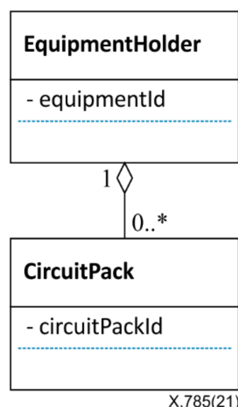

Figure II.1 – An example of containment relationship in UML

(2) An example for the usage of AssociationRelationshipType given in clause 8.2.5

For example, there is an association relationship between MOC Trail and MOC LinkConnection (see [ITU-T M.3160]), a JSON instance of AssociationRelationshipType may be provided as shown in Table II.2:

Table II.2 – An example of AssociationRelationshipType

```
{
     "associationRelationshipName": "Trail-LinkConnection-Association"
     "associationDirection": "bidirectional"
     "fromClass: "Trail"
     "fromAssociationAttribute": "clientLinkConneciotnPointerList"
     "fromMuitiplicity": "zero_to_n"
     "toClass": "LinkConnection"
     "toAssociationAttribute": "serverTrailList"
     "toMuitiplicity": "zero_to_n"
}
```

The above example can be explained like this: zero or multiple instances of MOC Trail are associated with zero or multiple instances of LinkConnection, and it is a bidirectional association, the name of the relationship is Trail-LinkConnection-Association. The association attribute from the Trail MOC to the LinkConnection MOC is clientLinkConneciotnPointerList, and the association attribute from the LinkConnection MOC to the Trail MOC is serverTrailList. This is equivalent to the UML diagram as shown in Figure II.2.

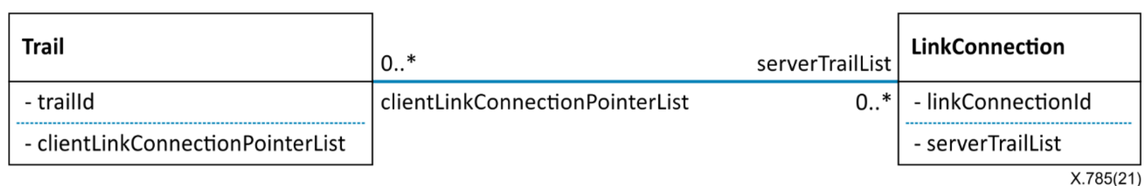

Figure II.2 – An example of association relationship in UML

The above examples should be used together with a resource model, which can provide extra information than just JSON object definitions, as an optional choice. They are not used to express the relationships between data instances but only used to express the relationship between classes in a resource model. Its role in expressing containment and association relationships is similar to that of UML, but in a JSON format, which may be resolved using the same tool for resolving REST interfaces.

Appendix III

Background for REST and HTTP technologies

(This appendix does not form an integral part of this Recommendation.)

III.1 Background

REST technology is now broadly used in the IT Industry. In some organizations and fora, research work has started on how to apply REST technology in the network management field as an alternative interface technology.

When using a REST technology in network management interfaces, some guidelines on how to use it to define interfaces and managed entities, as well as some supporting services should be provided. These guidelines, supporting services and some common definitions of generic managed objects together can be called the framework for REST-based paradigm.

The purpose of this document is to provide some related information in order to establish the framework for defining REST based network management interfaces and supporting services, so that in the future, specific REST-based interface definitions can follow those guidelines, and reuse some common services.

III.2 Short review of REST and HTTP

III.2.1 REST design principles

REST stands for representational state transfer. It is an architectural style defined by the following principles:

1) Client-server architecture

REST follows a client-server architecture. Client and server are linked by the uniform interface. The server is concerned with data storage. The client manipulates this data with create, read, update and delete (CRUD) operations. This architecture allows the client and server to evolve independently.

2) Stateless servers

REST servers are stateless, meaning that no client context is stored on the server. It is the client holding the session state. Each request from a client contains all the information required to service the request.

3) Cacheability

REST is cacheable. The client and any intermediary can cache responses, helping to improve system scalability and performance.

4) Layered system

REST is a **layered system**. A client cannot know if it is interacting with the end server or an intermediate server on the way to the end server. Each component has only knowledge about the component it is interacting with. All components are independent and easily replaceable or extendable. This improves system scalability and enables load-balancing.

5) Code on demand

Code on demand is an optional REST feature. It allows servers to transfer executable code to the client, thereby extending the functionality of the client.

6) Uniform interface

The uniform interface is the most important aspect of REST. Client and server communicate via the uniform interface. It is characterized by the following:

– *Resource identification*: The key concept is to abstract information into resources. These resources have a unique resource identification. Requests are directed towards resources.

– *Resource representation*: Each resource has one or multiple representations. Representations can be in e.g., XML, JSON or HTML. Resource representations are exchanged over the wire together with any representation metadata. The metadata provides information about the representation, such as its media type, the date of last modification, or even a checksum.

– *Self-descriptive messages*: Messages must be self-descriptive. All the information required to process the message is included in the message.

– *Hypermedia as the engine of application state* (HATEOAS): This refers to the capability of the server to send hyperlinks to the client allowing the client to traverse and dynamically discover resources without referring to external documentation.

III.2.2 HTTP methods

HTTP has several methods that can be used for this Recommendation, which are listed in Table III.1.

Table III.1 – Introduction of HTTP methods to be used

HTTP methods	Explanations
HTTP GET	The HTTP GET method requests a representation of the resource specified by the URI. It is used to retrieve one or multiple resources from the server. The query component of the URI can be used for filtering purposes in case more than one resource is scoped by the path-abempty part of the URI. Only those resources passing the filtering criteria are returned.
HTTP HEAD	The HTTP HEAD method returns only the headers that are returned with a HTTP GET method together with the message body, except for the payload header fields. This method can be used to check if resources exist.
HTTP POST	The POST method sends data in the message body to the server. In contrast to HTTP PUT, replacing the resource representation, it requests the target resource to process the representation enclosed in the request according to the resource's own specific semantics. With this method, it is possible to create a new resource. When a new resource is created, 201 (Created) is returned. The returned location header carries the URI of the created resource. The URI of the new resource is created by the server. The response message body contains a representation of the created resource.
HTTP PUT	The HTTP PUT method requests that the resource representation of the target resource be created or replaced with the representation enclosed in the request message payload. This method replaces always the complete resource representation. Partial resource modifications are not possible. If a resource at the URI specified in the request does not exist yet, the server creates a new resource at this URI. Conditional requests ([RFC 7232]) using e.g., the entity tag (ETag) can be used to prevent accidentally overwriting modifications made to a resource by another client ("lost update problem").
HTTP DELETE	The DELETE method requests that the origin server deletes the resource identified by the Request-URI. This does not imply that the underlying information is deleted as well.

Table III.1 – Introduction of HTTP methods to be used

HTTP methods	Explanations
HTTP PATCH	The HTTP PUT method only allows a complete resource replacement. For this reason, a new method, HTTP PATCH, has been defined by IETF in [RFC 5789] for partial resource modifications. The set of changes to be applied is described in the request message body.

HTTP has already provided several methods to carry the interaction capabilities between managing and managed systems.

III.3 Benefits of introducing REST into network management domain

REST provides a simplified mechanism to connect applications regardless of the technology or devices they use, or their location. They are based on industry standard protocols with universal vendor support that can leverage the internet for low cost communications, as well as other transport mechanisms. The loosely coupled messaging approach supports multiple connectivity and information sharing scenarios via services that are self describing and can be automatically discovered.

REST solutions uses HTTP as its operation protocols, which have been broadly used in the IT-industry for years, and it is mature and cost effective. The following features can be made use of when it is applied in the network management domain.

1) Good interoperability

REST solution has good support for it is universally interoperable, as far as the application supports the globally used protocol HTTP, it can be connected to the REST environment.

2) Loosely coupled

Loosely coupled systems require a much simpler level of coordination and allow for more flexible reconfiguration, compared to tightly coupled systems.

REST solutions are self-describing software modules that encapsulates discrete functionality. REST-based services are accessible via standard internet communication protocol HTTP directly. These services can be developed in any technology (like C++, Java, .NET, PHP, Pearl etc.) and any application can access these services. So, the REST-based services are loosely coupled and can be used by applications developed in any technology.

3) Broadly used

With the more rapid development of internet-based technologies, REST-based services are now broadly used in the IT service industries, for example e-business, business-to-business applications.

4) Low cost

REST APIs are open standards, and many tools, products, technologies are based on HTTP applications. This gives organizations a wide variety of choices, and they can select configurations that best meet their application requirements. Developers can enhance their productivity with low cost, rather than having to develop their own solutions, they can choose from a ready market of off-the-shelf application components or third-party tools.

Bibliography

[b-OAI-OAS3] OAI-OAS3 (2017), OpenAPI Initiative, *OpenAPI Specification Version 3.0.0.* http://spec.openapis.org/oas/v3.0.0

[b-3GPP TS 32.158] 3GPP TS 32.158 (2020), *Management and orchestration; Design rules for REpresentational State Transfer (REST) Solution Sets (SS) (Release 16), Version 16.2.0.* https://www.3gpp.org/ftp/Specs/archive/32_series/32.158/32158-g20.zip

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	Tariff and accounting principles and international telecommunication/ICT economic and policy issues
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Environment and ICTs, climate change, e-waste, energy efficiency; construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling, and associated measurements and tests
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	**Data networks, open system communications and security**
Series Y	Global information infrastructure, Internet protocol aspects, next-generation networks, Internet of Things and smart cities
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2021