INTERNATIONAL TELECOMMUNICATION UNION

TELECOMMUNICATION STANDARDIZATION SECTOR

STUDY PERIOD 2017-2020

SG2-C161

STUDY GROUP 2

Original: English

Question(s): Q7/2 Geneva, 19-28 February 2019

CONTRIBUTION

Source: Beijing University of Posts and Telecommunications

Title: Draft X.rest: Guidelines for defining REST-based managed objects and management interfaces

Purpose: Proposal

Contact: WANG Zhili
BUPT
China
Tel: +86 10 61198090 ext. 8726
Fax: +86 10 62283412
Email: zlwang@bupt.edu.cn

Contact: YU Peng
BUPT
China
Tel: +86 10 61198098 ext. 8710
Fax: +86 10 62283412
Email: yupeng@bupt.edu.cn

Contact: QIU Xuesong
BUPT
China
Tel: +86 10 61198065
Fax: +86 10 62283412
Email: xsqiu@bupt.edu.cn

Contact: LI Wenjing
BUPT
China
Tel: +86 10 61198090 ext. 8501
Fax: +86 10 62283412
Email: wjli@bupt.edu.cn

Contact: LIN Wei
Inspur
China
Tel: +86 10 56701901
Fax: +86 10 56701818
Email: lin.wei@inspur.com

Keywords: Distributed processing; (REST), managed objects, network management interfaces, JSON, JSON Schema.

Abstract: This document defines a set of guidelines for managed object modelling and a management interface for REST-based network management. It composes a framework for REST-based network management interfaces along with draft Q.rest. It specifies how REST-based management interfaces should be defined. It covers generic accessing methods of XML-based managed objects, information modelling in REST/HTTP and JSON schema. Some HTTP requests/responses and JSON schema are provided for defining some basic data types: generic managed object (MO) and generic MO accessing methods. This document and draft Q.rest together compose a framework for REST-based network management interfaces with a wide range of applications.

Draft Recommendation ITU-T X.rest:

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 GDMO/CMIP, CORBA GIOP/IIOP, SMI/SNMP, Web Service/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/SOAP is now an additional paradigm for network management.

This draft Recommendation, together with [ITU-T Q.rest] sets out to define a framework for defining how interfaces supported by management systems and network elements should be modelled using REST/JSON schema. It is within the scope of this Recommendation to provide the following guidelines or instructions:

– principles for REST interface definitions;

– containment relationship and naming rules for managed entities;

– generic accessing methods for managed objects;

– inheritance of managed objects and interfaces;

– information modelling guidelines for REST/JSON based interfaces;

– style conventions for REST/HTTP and JSON schema specifications;

– common data type and exception 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 (2011), *Management interface specification methodology*.

[ITU-T M.3160] Recommendation ITU-T M.3160 (2008),*Generic Network Management Information model - protocol neutral.*

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

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

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

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

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

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

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

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

Editor's Note: to be extended.

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:

BNF	Backus-Naur Form
CMIP	Common Management Information Protocol
CORBA	Common Object Request Broker Architecture
GDMO	Guidelines for the Definition of Managed Objects
GIOP	General Inter-ORB Protocol
HTTP	Hyper Text Transfer Protocol
IIOP	Internet Inter-ORB Protocol
JSON	JavaScript Object Notation
REST	REpresentational State Transfer
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
XSD	XML Schema Definition

Editor's Note: to be extended.

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

```
{
    "title": "root",
    "items": {
        "title": "array item"
    }
}
```

6 Overview of a REST-based management framework

REST-based technologies have been widely used in the IT industry. Appendix I provides more information on the features of web services technology. REST technology is similar to Web Services technology, and can be used in network management interfaces.

This Recommendation together with [ITU-T Q.rest] sets up a framework for defining how interfaces supported by management systems and network elements should be modelled using REST APIs and JSON schema.

The REST-based management framework include the following aspects:

1) Managed object and interface definition guidelines:
   - definition of managed objects using JSON schema;
   - accessing methods for MOs;
   - inheritance of MOs and interface operations;
   - information modelling guidelines for REST-based interface operations;
   - style idioms for JSON schema specifications.
2) REST supporting services for network management:
   - definition of a REST-based notification services;
   - definition of a REST-based heartbeat service;
   - definition of a REST-based multiple object operations (MOO) service;
   - definition of a REST-based containment service.

This Recommendation mainly deals with the managed object and interface definition guidelines, and draft [ITU-T Q.rest] mainly deals with the REST-based supporting services for network management. The two Recommendations together form a REST-based management framework.

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 and their accessing methods.

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

This Recommendation, along with [ITU-T Q.rest], defines a lightweight generic use of REST-based interface design patterns. The management and controlling functions are defined using HTTP methods, not 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 MOC is defined as an resource when exchanged through the management interface, and each attribute or state of the MOC is defined as an element in the resource.

– The naming of MOC instances follows the concept of a URI, which can be accessed using a HTTP request.

– There are four basic accessing methods for managed objects in the traditional TMN management paradigm, which are: createMO, deleteMO, getMOAttributes, 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 an specific resource.

– Common data types are defined in JSON schema which can be shared by application-specific interface definitions.

– Notifications sent from the agent to the manager should follow the format and behaviour defined in [ITU-T Q.rest]. The control of notification management services are also defined in [ITU-T Q.rest].

8 Definition of a generic managed object using JSON 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 funcare defined to provide interfaces to manage MOs.

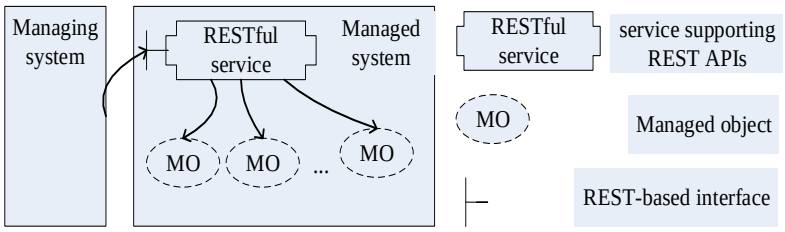

Figure 1 – RESTful services role

Figure 1 shows how a managing system accesses a managed system that supports a web services interface. A web 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 schema

An MO is the OSI management view of a resource that is subject to management, such as a connection or an item of physical equipment. Thus, an MO is the abstraction of such a resource that represents its properties for the purpose of management. An MO may 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, state can be described by JSON schema and behaviour by REST APIs. One important benefit of using an JSON document to store an MO's state is that REST APIs can also use JSON schema to describe the data type of its exchanged messages, and these JSON-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 super class, namely a ManagedObject. Using ManagedObject to define new MOCs will be easier and faster and provide better maintenance. As mentioned above, all MOCs are described in JSON schema and the data type of ManagedObject is given in Table 1 and the attributes can be found in Table 2.

Table 1 – Data type of super class ManagedObject

```
ManagedObject:
  type: object
  Required: :
    - objectClass
    - objectInstance
    properties:
      objectClass:
        type: string
      objectInstance:
        type: string
        Format: uri
      creationSource:
        $ref: '#/definitions/SourceIndicator'
```

```
SourceIndicator:
   type string
   enum:
     - ResourceOperation
     - ManagementOperation
     - unknown
```

Table 2 – Attributes of super class ManagedObject

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

As shown in Table 2, ManagedObject is made up of three attributes including objectClass, objectInstance, and creationSource. An attribute has an associated value with an 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 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. Resources in REST include document resources, collection resources, and task resources. The object instance is named URI string conforming to the following BNF paradigm specification.

```
    URI = {URI-prefix}/{ResourcePath}

    URI-prefix = {irpRoot}/{irpName}/{irpVersion}

    ResourcePath={DocumentResourcePath}|{CollectionResourcePath}|{TaskR
esourcePath}

    DocumentResourcePath = *( "/" RDN )

    CollectionResourcePath=*("/"RDN)"/"{namingAttributeName}

    TaskResourcePath=*("/"RDN) ["/" {namingAttribute}] "/" {actionName}

    RDN={namingAttributeName} "=" {namingAttributeValue}
```

(1)

Commented [WZL1]: Editor's Note: To be confirmed.

The attributes in the above formula should be the naming attribute of an MOC.

A complete JSON schema definition for the generic ManagedObject is defined in clause A.1.

(R) OBJECT-1. All the classes used to model resources on a managed system shall inherit (directly or indirectly) from the *ManagedObject* described above and defined in the JSON 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 base class, and extend their own unique attribute definitions based on public attributes to more clearly express their own feature information . In JSON 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.3100], the related object classes and the inheritance relationship between each object class can be shown in Figure 2.

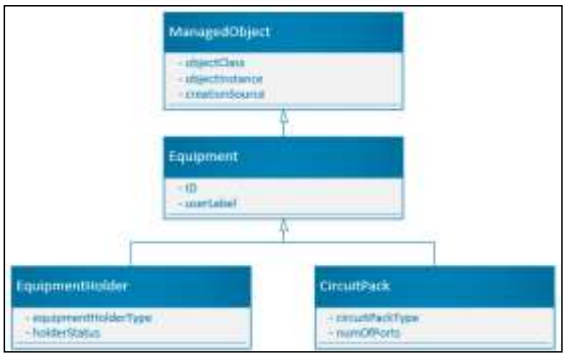

Figure 2 – An example for inheritance relationship

Table 3 – Data type of Equipment and EquipmentHolder by extension

```
Equipment:
  allOf:
     - $ref: '#/definitions/Managedobject'
     - properties:
         ID:
           type: integer
         userLabel:
           type: string
     - required:
        - ID
        - userLabel
```

```
EquipmentHolder:
  allOf:
     - $ref: '#/definitions/Equipment'
     - properties:
         equiepmentHolderType:
           type: string
         holderStatus:
           $ref: "#/definitions/holderStatusType'
     - required:
        - equiepmentHolderType
        - holderStatus
```

Some managed objects may need to support multiple inheritance, and the “allOf” syntax of JSON Schema itself supports multiple inheritance. In addition, JSON 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 JSON definitions for the above data types can be found in clause 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 include several racks, and one rack may contain several chassis. A chassis can contain a number of slots, and a board that performs various functions is inserted into the slot. There may be various ports on the board. In a containment relationship, an object class (or object instance) used to contain other managed objects is called a superior, and an included 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 JSON Schema, which defines five attributes including the relationship name, the parent class name, the parent class multiplicity, the subclass name, the subclass multiplicity, and the named property, and is shown in Figure 5.

Figure 3 – The JSON Schema for ContainmentRelationshipType

8.2.5 Association relationship of managed objects

In addition to the inheritance and containment relationships, there are also association relationship 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 JSON Schema of AssociationRelationshipType is defined. The attributes of the association relationship include: association name, association direction, from association class name, from association class attribute name, from association multiplicity, to association class name, to association class attribute name, to association multiplicity. The types of the attributes are all string type. The JSON Schema definition of the association type is shown in Figure 4.

Figure 4 – The JSON Schema for AssociationRelationshipType

Annex A

Common REST-based JSON schema definitions

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

A.1 JSON schema definitions for common data types and a generic managed object

Editor's Note: to be extended.

```
ManagedObject:
  type: object
  Required: :
    - objectClass
    - objectInstance
    properties:
      objectClass:
        type: string
      objectInstance:
        type: string
        Format: uri
      creationSource:
        $ref: '#/definitions/SourceIndicator'
SourceIndicator:
   type string
   enum:
     - ResourceOperation
     - ManagementOperation
     - unknown
```

Appendix I

Background for REST and HTTP technologies

I.1 Background

REST technology is now broadly used in IT Industry. In some organization and fora, they have started the research work on how to apply REST technology in 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 defined interface 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 trying to provide some related information in order to establish the framework for defining REST based network management interface and supporting services, so that in the future, specific REST-based interface definition can follow those guidelines, and reuse some common services.

I.2 Short review of REST and HTTP

I.2.1 REST design principles

REST stands for **RE**presentational **S**tate **T**ransfer. 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.

I.2.2 HTTP methods

HTTP has several methods can be used, which are listed in Table 1:

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.
HTTP CONNECT	
HTTP OPTIONS	
HTTP TRACE	
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 have already provided several methods to carry the interaction capabilities between managing and managed systems.

I.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 a good support for is universally interoperable, as far as the application support 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 which encapsulates discrete functionality. REST-based services are accessible via standard Internet communication protocol HTTP directly. These services can be developed in any technologies (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 technologies.

3) Broadly used

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

4) Low cost

REST APIs are open standards, and many tools, products, technologies is 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 because 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.