TM Forum Specification

TMF630 REST API Design Guidelines Part 4

Advanced guidelines for RESTful APIs lifecycle management, common tasks

TMF630

Team Approved Date: Nov. 25th, 2020

Release Status: Pre-Production	**Approval Status:** Team Approved
Version: 4.0.0	**IPR Mode:** RAND

NOTICE

Direct inquiries to the TM Forum office:

181 New Road, Suite 304
Parsippany, NJ 07054, USA
Tel No. +1 973 944 5100
Fax No. +1 973 998 7916
TM Forum Web Page: www.tmforum.org

Table of Contents

Executive Summary

This document, “REST API Design Guidelines Part 4” provides information for the development of TM Forum APIs using REST.

It provides recommendations and guidelines for the implementation of the Common Export and Import Tasks and the Entity Lifecycle Management related patterns.

Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Chapter 1. Export and Import Data Tasks

This section describes the common tasks used for exporting and importing resources representations to files.

Two common task resources are defined: * ImportJob used to import resources from a File * ExportJob used to export resources to a File

1.1. Export Job Resource

An ExportJob resource represents a TASK used to export resources to a File

The ExportJob resource supports the following properties:

Attribute name	Description
query	Used to scope the exported data (identical to GET filter construct using target ID as base) “query”: ”type=productOffering&version=2.0”
path	URL of the root resource acting as the source for for streaming content to the file specified by the ExportJob ../catalogManagement/catalog/42 i.e the relative path to the exportJob when it was created can also be used
content-type	The format of the exported data .By default “application/json”
status	notstarted, running, succeeded, failed
url	URL of the File containing the data to be exported a file URL, which is of the form file://host/path where host is the fully qualified domain name of the system on which the path is accessible, and path is a hierarchical directory path of the form directory/directory/…/name

Attribute name	Description
completionDate	Date at which the Job was completed.
creationDate	Date at which the Job was created.
errorLog	Reason for Failure

1.2. Import Job Resource

An ImportJob resource represent a TASK used to import resources from a File

The ImportJob resource supports the following properties:

Attribute name	Description
content-type	The format of the imported data .By default `application/json`
path	URL of the root resource where the content of the file specified by the ImportJob must be applied ../catalogManagement/catalog/42 i.e the relative path to the importJob when it was created
status	notstarted, running, succeeded, failed
url	URL of the File containing the data to be imported a file URL, which is of the form file://host/path where host is the fully qualified domain name of the system on which the path is accessible, and path is a hierarchical directory path of the form directory/directory/…/name
completionDate	Date at which the Job was completed.
creationDate	Date at which the Job was created.
errorLog	Reason for Failure if status is failed

1.3. Creating Export Jobs

ExportJob Tasks are created as resources. The ExportJob is attached to a specific resource acting as the root for the collection of resources to be streamed to a File.

An ExportJob can be attached to a specific Catalog in a Catalog application or may be attached to the Product Offerings within a Catalog.

- ../catalogManagement/catalog/42/exportJob

  Export all the resources within the Catalog 42

- ../catalogManagement/catalog/42/productOffering/exportJob

  Export all the ProductOffering resources

  For example:

REQUEST

```
POST catalogManagement/catalog/\{10}/exportJob
Content-type: application/json
{
}
```

RESPONSE

```
201
Content-Type: application/json
Location: /api/catalogManagement/exportJob/54

{
    id": "54",
    "href": "/catalogManagement/exportJob/54",
    "status": "running",
    path: catalogManagement/catalog/10" ,
    content-type: application/json,
    "errorLog": "",
    creationDate : 2013-04-19T16:42:23-04:00",
    completionDate : ",
    "url": "ftp://ftp.myCatalog.com/productCatalog/54"
}
```

1.4. Creating Import Jobs

ImportJob Tasks are created as resources. The ImportJob may be attached to the URL of the root resource where the content of the file specified by the ImportJob will be applied.

For example, to apply the content of the import file to the catalog 10:

REQUEST

```
POST catalogManagement/catalog/\{10}/importJob

Content-type: application/json

{
}
```

RESPONSE

```
201
Content-Type: application/json
Location: /api/catalogManagement/importJob/554

{
    "id": "554",
    "href": "/catalogManagement/importJob/554",
    "status": "running",
    "path": "catalogManagement/catalog/10" ,
    "content-type": "application/json",
    "errorLog": "",
    "creationDate" : "2013-04-19T16:42:23-04:00",
    "completionDate" : "",
    "url": "ftp://ftp.myCatalog.com/productCatalog/partner54"
}
```

1.5. Query Export Jobs

ExportJob resources can be found under the API/exportJob collection and may be retrieved using the normal GET constructs.

For example:

REQUEST

```
GET API/catalogManagement/exportJob/\{10}
Accept: application/json
```

RESPONSE

{ "id": "10", "status": "Succeeded" }

1.6. Query Import Jobs

ImportJob resources can be found under the API/importJob collection and may be retrieved using the normal GET constructs.

REQUEST

```
GET /catalogManagement/importJob/\{25}
Accept: application/json
```

RESPONSE

```
{ +
"id": "25", +
"status": "Succeeded"
}
```

1.7. Export Job Completion Notification

This event provides notification that an export task has been completed.

For example:

REQUEST

```
POST /client/listener
Accept: application/json

{
    "event": {
        "id": "54",
        "href": "/catalogManagement/exportJob/54",
        "status": "running",
        "path": "catalogManagement/catalog/{10}" ,
        "content-type": "application/json",
        "errorLog": "",
        "creationDate" : "2013-04-19T16:42:23-04:00",
        "completionDate" : "", +
        "url": "ftp://ftp.myCatalog.com/productCatalog/54"
    },
    "eventType": "exportJobCompleted"
}
```

1.8. Import Job Completion Notification

This event provides a notification that an import task has been completed.

REQUEST

```
POST /client/listener
Accept: application/json

{
    "event": {
    "id": "554",
    "href": "/catalogManagement/importJob/554",
    "status": "running",
    "path": "catalogManagement/catalog/10" ,
    "content-type": "application/json",
    "errorLog": "",
    "creationDate" : "2013-04-19T16:42:23-04:00",
    "completionDate" : "2013-04-19T16:42:23-04:15",
    "url": "ftp://ftp.myCatalog.com/productCatalog/partner54"
    },
    "eventType": "importJobCompleted"
}
```

Chapter 2. Entity Versioning and Lifecycle Management

In Product Lifecycle Management there is a requirement to distinguish between entities existing with different PLM version numbers and accessible via different ACL mechanisms.

For example, the same Product Offerings may exist in a Catalog but with different version numbers.

It may be possible for an administrator to see all the existing versions or for a partner to see only a subset of all the existing versions.

The entity version number is not dependent on the version number of the API. For example, in PLM the same API (running at a specific version number) may be used to retrieve entities with different PLM version numbers.

In order to distinguish resources representing entities running with different version numbers and accessible though the same API version the following directive can be used /id:(version=x) and the version attribute is added to each entity.

```
{
    "id": "42",
    "href": "/catalogManagement/productOffering/42",
    "version": "1.0",
    "lastUpdate": "2013-04-19T16:42:23-04:00",
    "name": "Virtual Storage Medium",
    "description": "Virtual Storage Medium",
    "isBundle": "true",
    "lifecycleStatus": "Active"
}
```

Note that the resources in this case may have the same ID but may be distinguished by the inclusion of the version number in their ID i.e /42:(version=1.0), /42:(version=2.0).

In the following examples we will assume that two versions of the VirtualStorage Product Offer exist in the Product Catalog an Inactive and Active version respectively version 1.0 and version 2.0.

2.1. Query all versioned resources

Admin user of Catalog have access to all the versions of the resources while non admin users have by default access to only the latest version of the entities in the Catalog.

Users with different roles may have access to different versions of the entities in the catalog. For example, user A may have access to only the version 1.0 of the entities while user B may have access to version 1.0 and version 2.0.

For example, the following request on the admin endpoint will return all the versioned resources matching a specific ID.

REQUEST

```
GET api/admin/catalogManagement/productOffering/?id=VirtualStorage
Accept: application/json
```

RESPONSE

```
200
Content-Type: application/json

[{
    "id": " VirtualStorage ",
    "href": "/catalogManagement/productOffering/VirtualStorage",
    "version": "1.0",
    "lastUpdate": "2013-04-19T16:42:23-04:00",
    "name": "Virtual Storage Medium",
    "description": "Virtual Storage Medium",
    "isBundle": "true",
    "lifecycleStatus": "InActive",

},
{
    "id": " VirtualStorage ",
    "href": "/catalogManagement/productOffering/ VirtualStorage ",
    "version": "2.0",
    "lastUpdate": "2013-04-19T16:42:23-04:00",
    "name": "Virtual Storage Medium",
    "description": "Virtual Storage Medium",
    "isBundle": "true",
    "lifecycleStatus": "Active",
}]
```

2.2. Query a specific versioned resource

In general, a non admin API user only has visibility to the latest version number or visibility to a subset of versioned resources.

It may be possible for an admin API user to retrieve a resource with a specific version number by using an ID and versioning filtering criteria.

REQUEST

```
GET api/admin/catalogManagement/productOffering/?id=VirtualStorage&version=1.0
Accept: application/json
```

RESPONSE

```
200
Content-Type: application/json

[{
    "id": "42",
    "href": "/catalogManagement/productOffering/42",
    "version": "1.0",
    "lastUpdate": "2013-04-19T16:42:23-04:00",
    "name": "Virtual Storage Medium",
    "description": "Virtual Storage Medium",
    "isBundle": "true",
    "lifecycleStatus": "Active"
}]
```

2.3. Query current version of a resource

By default, only the most current version is returned (for admin and non admin).

REQUEST

```
GET api/admin/catalogManagement/productOffering/VirtualStorage
Accept: application/json
```

RESPONSE

```
200
Content-Type: application/json

{
    "id": "42",
    "href": "/catalogManagement/productOffering/42",
    "version": "2.0",
    "lastUpdate": "2013-04-19T16:42:23-04:00",
    "name": "Virtual Storage Medium",
    "description": "Virtual Storage Medium",
    "isBundle": "true",
    "lifecycleStatus": "Active"
}
```

2.4. Create new version of a resource

POST is used to create a new version of a resource.

The constraint is that the version numbers for the resource having the same ID must differ.

REQUEST

```
POST catalogManagement/productOffering

Content-type: application/json

{
    "id": "VirtualStorage",
    "version": "3.0",
    "name": "Virtual Storage Medium",
    "description": "Virtual Storage Medium",
    "isBundle": "true",
    "lifecycleStatus": "Active",
    "validFor": {
        "startDateTime": "2013-04-19T16:42:23-04:00",
        "endDateTime": "2013-06-19T00:00:00-04:00"
    }
}
```

RESPONSE

```
201

Content-Type: application/json

{
    "id": "VirtualStorage",
    "href": "/catalogManagement/productOffering/42",
    "version": "3.0",
    "lastUpdate": "2013-04-19T16:42:23-04:00",
    "name": "Virtual Storage Medium",
    "description": "Virtual Storage Medium",
    "isBundle": "true",
    "lifecycleStatus": "Active",
    "validFor": {
        "startDateTime": "2013-04-19T16:42:23-04:00",
        "endDateTime": "2013-06-19T00:00:00-04:00"
    },
}
```

2.5. Modify an existing version of a resource

By default, PATCH or PUT will be acting only on the latest version of the Resource. For example, PATCH /…/productOffering/VirtualStorage will only update the VirtualStoage ProductOffering at Version 2.0 (which is the most current).

To update a specific version of an entity the (Version=X) directive is added to the ID (i.e /id:(version=x).

Note that this capability is only available to API users having the proper authorizations to change the catalog entities with specific version numbers.

For example, to change the VirtualStorage versioned at 1.0 we could use /productOffering/VirtualStorage(Version=1.0)

REQUEST

```
PATCH /catalogManagement/productOffering/VirtualStorage(Version=1.0)
Content-type: application/json-patch+json

{
    "lifecycleStatus": "Active"
    ...
}
```

RESPONSE

```
201

Content-Type: application/json

{
    "id": "VirtualStorage",
    "href": "/catalogManagement/productOffering/VirtualStorage",
    "version": "1.0",
    "lifecycleStatus": "Active",
    ...
}
```

2.6. Role based Access Control

The user presents their credentials for authentication

If the credentials are valid

1. The user is given access to the catalog

2. As defined by their role(s)

3. As defined by their access rights

4. As defined by the access type: CRUD, discover

5. As defined by the pre-defined filter

For example, if they issue a get on a catalog that a party has no access they get an error response

Or if they try to modify an area of the catalog but do not have Write Access they get an error response

We anticipate that the OAUTH2 or Open ID Connect will be used as the authorization APIs and that ACL are established between authorized parties with regards to the content of the Catalog (i.e., GET but also enable of update operations on specific entities).

Chapter 3. Administrative Appendix

This Appendix provides additional background material about the TM Forum and this document. In general, sections may be included or omitted as desired; however, a Document History must always be included.

3.1. Document History

3.1.1. Version History

This section records the changes between this and the previous document version as it is edited by the team concerned. Note: this is an incremental number which does not have to match the release number and used for change control purposes only.

Version Number	Date Modified	Modified by:	Description of changes
1.0	23-Nov-2014	Pierre Gauthier TM Forum	Description e.g. first issue of document
1.1.1	12/03/2015	Alicja Kawecki, TM Forum	Updated cover, footer and Notice to reflect TM Forum Approved status
2.0.0	08-Nov-2017	Peter Norbury	Update to Notice and changed document number
2.0.1	12-Dec-2017	Adrienne Walcott	Formatting/style edits prior to publishing
2.0.2	20 Mar 2018	Adrienne Walcott	Updated to reflect TM Forum Approved Status

3.2. Release History

This section records the changes between this and the previous Official document release. The release number is the 'Marketing' number which this version of the document is first being assigned to.

Release Number	Date Modified	Modified by:	Description of changes
17.5.0	08-Nov-2017	Peter Norbury	first release of document
17.5.1	20-Mar-2018	Adrienne Walcott	Updated to reflect TM Forum Approved Status

3.3. Acknowledgments

This document was prepared by the members of the TM Forum [team name] team:

- Pierre Gauthier, TM Forum, **Editor** and Team Leader
- Maxime Delon Orange
- Veronique Mauneau Orange
- John Morey Ciena