REST Resource Naming Guide

Last Updated : November 23, 2021 By : Lokesh Gupta

1. What is a Resource?

In REST, the primary data representation is called **resource**. Having a consistent and robust REST resource naming strategy – will prove one of the best design decisions in the long term.

> The key abstraction of information in REST is a resource. Any information that can be named can be a resource: a document or image, a temporal service (e.g. “today’s weather in Los Angeles”), a collection of other resources, a non-virtual object (e.g., a person), and so on.
>
> In other words, any concept that might be the target of an author’s hypertext reference must fit within the definition of a resource.
>
> A resource is a conceptual mapping to a set of entities, not the entity that corresponds to the mapping at any particular point in time.
>
> — *Roy Fielding’s dissertation*

1.1. Singleton and Collection Resources

A **resource can be a singleton or a collection**.

For example, “`customers`” is a collection resource and “`customer`” is a singleton resource (in a banking domain).

We can identify “`customers`” collection resource using the URI “`/customers`“. We can identify a single “`customer`” resource using the URI “`/customers/{customerId}`“.

1.2. Collection and Sub-collection Resources

A **resource may contain sub-collection resources** also.

For example, sub-collection resource “`accounts`” of a particular “`customer`” can be identified using the URN “`/customers/{customerId}/accounts`” (in a banking domain).

Similarly, a singleton resource “`account`” inside the sub-collection resource “`accounts`” can be identified as follows: “`/customers/{customerId}/accounts/{accountId}`“.

1.3. URI

REST APIs use Uniform Resource Identifiers (URIs) to address resources. REST API designers should create URIs that convey a REST API’s resource model to the potential clients of the API. When resources are named well, an API is intuitive and easy to use. If done poorly, that same API can be challenging to use and understand.

> *The constraint of a uniform interface is partially addressed by the combination of URIs and HTTP verbs and using them in line with the standards and conventions.*

Below are a few tips to get you going when creating the resource URIs for your new API.

2. Best Practices

2.1. Use nouns to represent resources

RESTful URI should refer to a resource that is a thing (noun) instead of referring to an action (verb) because nouns have properties that verbs do not have – similarly, resources have attributes. Some examples of a resource are:

- Users of the system
- User Accounts
- Network Devices etc.

and their resource URIs can be designed as below:

```
http://api.example.com/device-management/managed-devices
http://api.example.com/device-management/managed-devices/{device-id}
http://api.example.com/user-management/users
http://api.example.com/user-management/users/{id}
```

For more clarity, let’s divide the **resource archetypes** into four categories (document, collection, store, and controller). Then it would be best if **you always targeted to put a resource into one archetype and then use its naming convention consistently**.

For uniformity’s sake, resist the temptation to design resources that are hybrids of more than one archetype.

2.1.1. document

A document resource is a singular concept that is akin to an object instance or database record.

In REST, you can view it as a single resource inside resource collection. A document’s state representation typically includes both fields with values and links to other related resources.

Use “singular” name to denote document resource archetype.

```
http://api.example.com/device-management/managed-devices/{device-id}
http://api.example.com/user-management/users/{id}
http://api.example.com/user-management/users/admin
```

2.1.2. collection

A collection resource is a server-managed directory of resources.

Clients may propose new resources to be added to a collection. However, it is up to the collection resource to choose to create a new resource or not.

A collection resource chooses what it wants to contain and also decides the URIs of each contained resource.

Use the “plural” name to denote the collection resource archetype.

```
http://api.example.com/device-management/managed-devices
http://api.example.com/user-management/users
```

```
http://api.example.com/user-management/users/{id}/accounts
```

2.1.3. store

A store is a client-managed resource repository. A store resource lets an API client put resources in, get them back out, and decide when to delete them.

A store never generates new URIs. Instead, each stored resource has a URI. The URI was chosen by a client when the resource initially put it into the store.

Use “plural” name to denote store resource archetype.

```
http://api.example.com/song-management/users/{id}/playlists
```

2.1.4. controller

A controller resource models a procedural concept. Controller resources are like executable functions, with parameters and return values, inputs, and outputs.

Use “verb” to denote controller archetype.

```
http://api.example.com/cart-management/users/{id}/cart/checkout
http://api.example.com/song-management/users/{id}/playlist/play
```

2.2. Consistency is the key

Use consistent resource naming conventions and URI formatting for minimum ambiguity and maximum readability and maintainability. You may implement the below design hints to achieve consistency:

2.2.1. Use forward slash (/) to indicate hierarchical relationships

The forward-slash (/) character is used in the path portion of the URI to indicate a hierarchical relationship between resources. e.g.

```
http://api.example.com/device-management
http://api.example.com/device-management/managed-devices
http://api.example.com/device-management/managed-devices/{id}
http://api.example.com/device-management/managed-devices/{id}/scripts
http://api.example.com/device-management/managed-devices/{id}/scripts/{id}
```

2.2.2. Do not use trailing forward slash (/) in URIs

As the last character within a URI's path, a forward slash (/) adds no semantic value and may confuse. It's better to drop it from the URI.

```
http://api.example.com/device-management/managed-devices/
http://api.example.com/device-management/managed-devices  /*This is much better
version*/
```

2.2.3. Use hyphens (-) to improve the readability of URIs

To make your URIs easy for people to scan and interpret, use the hyphen (-) character to improve the readability of names in long path segments.

```
http://api.example.com/device-management/managed-devices/
http://api.example.com/device-management/managed-devices     /*This is much better
version*/
```

2.2.4. Do not use underscores (_)

It's possible to use an underscore in place of a hyphen to be used as a separator – But depending on the application's font, it is possible that the underscore (_) character can either get partially obscured or completely hidden in some browsers or screens.

To avoid this confusion, use hyphens (-) instead of underscores (_).

```
http://api.example.com/inventory-management/managed-entities/{id}/install-script-location
//More readable

http://api.example.com/inventory-management/managedEntities/{id}/installScriptLocation
//Less readable
```

2.2.5. Use lowercase letters in URIs

When convenient, lowercase letters should be consistently preferred in URI paths.

```
http://api.example.org/my-folder/my-doc       //1
HTTP://API.EXAMPLE.ORG/my-folder/my-doc     //2
http://api.example.org/My-Folder/my-doc       //3
```

In the above examples, 1 and 2 are the same but 3 is not as it uses **My-Folder** in capital letters.

2.3. Do not use file extensions

File extensions look bad and do not add any advantage. Removing them decreases the length of URIs as well. No reason to keep them.

Apart from the above reason, if you want to highlight the media type of API using file extension, then you should rely on the media type, as communicated through the `Content-Type` header, to determine how to process the body's content.

```
http://api.example.com/device-management/managed-devices.xml  /*Do not use it*/

http://api.example.com/device-management/managed-devices     /*This is correct URI*/
```

2.4. Never use CRUD function names in URIs

We should not use URIs to indicate a CRUD function. URIs should only be used to uniquely identify the resources and not any action upon them.

We should use HTTP request methods to indicate which CRUD function is performed.

```
HTTP GET http://api.example.com/device-management/managed-devices  //Get all devices
HTTP POST http://api.example.com/device-management/managed-devices  //Create new Device

HTTP GET http://api.example.com/device-management/managed-devices/{id}  //Get device for given Id
HTTP PUT http://api.example.com/device-management/managed-devices/{id}  //Update device for given Id
HTTP DELETE http://api.example.com/device-management/managed-devices/{id}  //Delete device for given Id
```

2.5. Use query component to filter URI collection

Often, you will encounter requirements where you will need a collection of resources sorted, filtered, or limited based on some specific resource attribute.

For this requirement, do not create new APIs – instead, enable sorting, filtering, and pagination capabilities in resource collection API and pass the input parameters as query parameters. e.g.

```
http://api.example.com/device-management/managed-devices
http://api.example.com/device-management/managed-devices?region=USA
http://api.example.com/device-management/managed-devices?region=USA&brand=XYZ
http://api.example.com/device-management/managed-devices?
region=USA&brand=XYZ&sort=installation-date
```