# API URLs

The structure of a URI is central to how APIs are organised and categorised within your enterprise domain. A good URI taxonomy helps to categorise your APIs across functional domains, regions, access (public or private) and helps define relationships (hierarchical). A good URI also helps to govern the lifecycle of your API through versioning practices.

Recommended URI Structure:

|  |  |  |
| --- | --- | --- |
| **Part** | **Description** | **Example** |
| {env} | **Optional**. The API environment. An API could be available in a sandbox environment to enable developers to test that API. The {env} part is excluded for production APIs. | sandbox |
| {access} | **Optional**. The access level of the API. This could be public or private. By default the {access} part is excluded for public APIs or simply set to "api". | api |
| {company} | Required. The name of the company or business division for private services. | mytaxis |
| {region} | Required. The region of the API | .[co.uk](http://co.uk) |
| {context} | Required. The name of the API as defined in the API Manager. This typically presents the business service and should be a short but descriptive name. | quickbooker |
| {version} | Required. The version of the API. Depending on requirements, the version can reflect only major versions or include a more hierarchical convention to identify minor versions. | v1 |
| {resource} | Required. The name of the resource that represents the actual object. An API may contain multiple resources. The resource can also be referred to as the API endpoint. | bookings |
| {resource-id} | Optional. The id of the resource to be fetched/updates. The resource id is optional. | 1981927 |
| {queryparams} | Optional. The query string can define state transition parameters. | page=1&sort=+<field> |

# Setting the Base URI

Once the APIs URLs are established, the baseURI defined in the RAML should reflect the DNS entires: http://[env].[access].[company].[region]/[context]/[version] as described in the table above.

# Filtering

In some cases, the API consumer might only need a subset of a collection of resources. This could be accomplished by using query parameters. For example, to get the list of all shipped orders, the API consumer could use:

GET /orders?state=shipped

Here, the "state" query parameter is used to filter the response.

# Sorting

Similar to filtering, a generic query parameter sort could be used to describe sorting rules. To allow sorting on multiple fields, the query parameter could be designed to take a list of fields instead of a single value.

Next, to allow for ascending and descending sort order, the query parameter could take minus (“-“) as a prefix of each field.

For example, the following request will return all purchase orders sorted by date (descending) and then by product (ascending):

GET /orders?sort=-date,product

# Partial Resources

In some cases, the consumer might not need all the fields of a resource. To allow for obtaining only a partial resource the API URL could be designed to take a list of fields as a query parameter, and return only the fields that are included in that list.

For example, the following request will return only the date and the total of the purchase order:

GET /orders/1?fields=date,total

## Aliases

To make the experience of using an API more pleasant for the application developers, the API could package a set of conditions into an easily accessible URL. For example, to return the recently shipped orders, the API could provide the following endpoint: GET /orders/most-recent

A resource name should remain short in order to avoid any size limitations.  The base URL should also contain no more than 2-3 resources if possible. URIs can be limited in some HTTP stacks.