**API Design Associate Raml 1.0**

* RAML Version 1.0: (RESTful API Modeling Language)----

RAML is a human- and machine-readable language for defining HTTP-based APIs which follows REST(Representational State Transfer) standards

RAML is a mutual contract between api provider and the api consumer

The RAML specification provides mechanisms for defining practically-RESTful APIs, creating client/server source code, and comprehensively documenting the APIs for users.

Earlier Version RAML V 0.8

RAML 1.0 supports YAML(Yet Another Markup Language) 1.2 as its underlying format.

#%RAML 1.0

title: My API

The first line of a RAML API definition document MUST begin with the text #%RAML followed by a single space followed by the text 1.0 and nothing else before the end of the line.

RAML fragment documents begin similarly with the RAML version comment and a [fragment identifier](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#typed-fragments)

* The first line of a RAML file MUST consist of a YAML comment that specifies the RAML version. Therefore, RAML processors SHALL NOT completely ignore all YAML comments.

## The Root of the Document

The root section of the RAML document describes the basic information about an API, such as its title and version. The root section also defines assets used elsewhere in the RAML document, such as types and traits.

This example shows a small part of a RAML API definition for the GitHub v3 public API.

#%RAML 1.0

title: GitHub API

version: v3

baseUri: https://api.github.com

mediaType: application/json

securitySchemes:

oauth\_2\_0: !include securitySchemes/oauth\_2\_0.raml

types:

Gist: !include types/gist.raml

Gists: !include types/gists.raml

resourceTypes:

collection: !include types/collection.raml

traits:

securedBy: [ oauth\_2\_0 ]

/users:

type: collection

get:

The following table enumerates the possible nodes at the root of a RAML document:

| **Name** | **Description** |
| --- | --- |
| title | A short, plain-text label for the API. Its value is a string. |
| description? | A substantial, human-friendly description of the API. Its value is a string and MAY be formatted using [markdown](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#markdown). |
| version? | The version of the API, for example "v1". Its value is a string. |
| baseUri? | A URI that serves as the [base for URIs](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#base-uri-and-base-uri-parameters) of all resources. Often used as the base of the URL of each resource containing the location of the API. Can be a [template URI](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#template-uri). |
| baseUriParameters? | Named parameters used in the [baseUri](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md" \l "base-uri-and-base-uri-parameters) (template). |
| protocols? | The [protocols](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#protocols) supported by the API. |
| mediaType? | The [default media types](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#default-media-types) to use for request and response bodies (payloads), for example "application/json". |
| documentation? | Additional overall [documentation](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#user-documentation) for the API. |
| schemas? | An alias for the equivalent "types" node for compatibility with RAML 0.8. Deprecated - API definitions SHOULD use the "types" node because a future RAML version might remove the "schemas" alias with that node. The "types" node supports XML and JSON schemas. |
| types? | Declarations of [(data) types](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#defining-types) for use within the API. |
| traits? | Declarations of [traits](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#resource-types-and-traits) for use within the API. |
| resourceTypes? | Declarations of [resource types](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#resource-types-and-traits) for use within the API. |
| annotationTypes? | Declarations of [annotation types](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#declaring-annotation-types) for use by annotations. |
| (<annotationName>)? | [Annotations](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#annotations) to be applied to this API. An annotation is a map having a key that begins with "(" and ends with ")" where the text enclosed in parentheses is the annotation name, and the value is an instance of that annotation. |
| securitySchemes? | Declarations of [security schemes](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#security-schemes) for use within the API. |
| securedBy? | The [security schemes](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#applying-security-schemes) that apply to every resource and method in the API. |
| uses? | Imported external [libraries](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#libraries) for use within the API. |
| /<relativeUri>? | The resources of the API, identified as relative URIs that begin with a slash (/). A [resource node](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#resources-and-nested-resources) is one that begins with the slash and is either at the root of the API definition or a child of a resource node. For example, /users and /{groupId}. |

We recommended using the "types" node instead of "schemas" because the schemas alias is deprecated and might be removed in a future RAML version.

### User Documentation

The OPTIONAL **documentation** node includes a variety of documents that serve as user guides and reference documentation for the API. Such documents can clarify how the API works or provide technical and business context.

The value of the documentation node MUST be a sequence of one or more documents. Each document is a map that MUST have exactly two key-value pairs described in the following table:

| **Name** | **Description** |
| --- | --- |
| title | Title of the document. Its value MUST be a non-empty string. |
| content | Content of the document. Its value MUST be a non-empty string and MAY be formatted using [markdown](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#markdown). |

This example shows an API definition having two user documents.

#%RAML 1.0

title: ZEncoder API

baseUri: https://app.zencoder.com/api

documentation:

- title: Home

content: |

Welcome to the \_Zencoder API\_ Documentation. The \_Zencoder API\_

allows you to connect your application to our encoding service

and encode videos without going through the web interface. You

may also benefit from one of our

[integration libraries](https://app.zencoder.com/docs/faq/basics/libraries)

for different languages.

- title: Legal

content: !include docs/legal.markdown

### Base URI and Base URI Parameters

The OPTIONAL **baseUri** node specifies a URI as an identifier for the API as a whole, and MAY be used to specify the URL at which the API is served (its service endpoint), and which forms the base of the URLs of each of its resources.

If the baseUri value is a [Template URI](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#template-uri), the following reserved base URI parameter is available.

| **URI Parameter** | **Value** |
| --- | --- |
| Version | The value of the root-level version node |

Any other URI template variables appearing in the baseUri MAY be described explicitly within a **baseUriParameters** node at the root of the API definition. The baseUriParameters node has the same semantics and therefore MUST follow the same structure as the [uriParameters](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md" \l "template-uris-and-uri-parameters) node on a resource node, except that it specifies parameters in the base URI rather than the relative URI of a resource.

The following example RAML API definition uses a [Template URI](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#template-uri) as the base URI.

#%RAML 1.0

title: Salesforce Chatter REST API

version: v28.0

baseUri: https://na1.salesforce.com/services/data/{version}/chatter

The following example declares an explicit base URI parameter.

#%RAML 1.0

title: Amazon S3 REST API

version: 1

baseUri: https://{bucketName}.s3.amazonaws.com

baseUriParameters:

bucketName:

description: The name of the bucket

When the base URI ends in one or more slashes (/), the trailing slashes are omitted in the absolute paths for the resources using that base URI. For example, in the following snippet, the absolute paths for the resources are http://api.test.com/common/users/:userId and http://api.test.com/common/users/:userId/groups.

baseUri: http://api.test.com/common/

/users:

/{userId}:

/groups:

### Protocols

The OPTIONAL **protocols** node specifies the protocols that an API supports. If the protocols node is not explicitly specified, one or more protocols included in the baseUri node SHALL be used; if the protocols node is explicitly specified, such node specification SHALL override any protocol included in the baseUri node. The protocols node MUST be a non-empty array of strings, of values HTTP and/or HTTPS, and be case-insensitive.

The following is an example of an API endpoint that accepts both HTTP and HTTPS requests.

#%RAML 1.0

title: Salesforce Chatter REST API

version: v28.0

protocols: [ HTTP, HTTPS ]

baseUri: https://na1.salesforce.com/services/data/{version}/chatter

### Default Media Types

Specifying the OPTIONAL **mediaType** node sets the default media type for responses and requests that have a body. You do not need to specify the media type within every body definition.

The value of the mediaType node MUST be a sequence of media type strings or a single media type string. The media type applies to requests having a body, the expected responses, and examples using the same sequence of media type strings.

This example shows a RAML snippet for an API that accepts and returns a JSON-formatted body. If the remainder of this API specification does not explicitly specify another media type, this API accepts and returns only JSON-formatted bodies.

#%RAML 1.0

title: New API

mediaType: application/json

This example shows a RAML snippet for an API that accepts and returns JSON- or XML-formatted bodies.

#%RAML 1.0

title: New API

mediaType: [ application/json, application/xml ]

Explicitly defining a mediaType node for a [body](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#bodies) of an API request or response overrides the default media type, as shown in the following example. The resource /people returns a Person[] body in either JSON or XML. The resource /messages overrides the default media type by explicitly defining an application/json node. Therefore, the resource /messages returns only a JSON-formatted body.

#%RAML 1.0

title: New API

mediaType: [ application/json, application/xml ]

types:

Person:

Another:

/people:

get:

responses:

200:

body: Person[]

/messages:

post:

body:

application/json:

type: Another

### Default Security

Specifying the OPTIONAL **securedBy** node sets the default security schemes for, and protects, every method of every resource in the API. The value of the node MUST be an array of security scheme names

The following example shows an API allowing access through either an OAuth 2.0 security scheme or an OAuth 1.1 security scheme.

#%RAML 1.0

title: Dropbox API

version: 1

baseUri: https://api.dropbox.com/{version}

securedBy: [ oauth\_2\_0, oauth\_1\_0 ]

securitySchemes:

oauth\_2\_0: !include securitySchemes/oauth\_2\_0.raml

oauth\_1\_0: !include securitySchemes/oauth\_1\_0.raml

## RAML Data Types

Data types can describe a base or resource URI parameter, a query parameter, a request or response header, or a request or response body. Data types are built-in or custom. A built-in type can be used anywhere the API expects data. Custom types can be defined by extending the built-in types as well as named and used like built-in type.

The following RAML example defines a User type that includes type declarations for the firstname, lastname, and age properties. The example declares the properties to be of built-in types string and number. Later, the User type serves to describe the type (schema) for a payload.

#%RAML 1.0

title: API with Types

types:

User:

type: object

properties:

firstname: string

lastname: string

age: number

/users/{id}:

get:

responses:

200:

body:

application/json:

type: User

### Built-in Types

The RAML type system defines the following built-in types:

* [any](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#the-any-type)
* [object](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#object-type)
* [array](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#array-type)
* [union](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#union-type) via type expression
* one of the following [scalar types](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#scalar-types): number, boolean, string, date-only, time-only, datetime-only, datetime, file, integer, or nil
* The following diagram shows the inheritance tree, starting at the root-level with any.

#### [Types Hierarchy](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/images/typesHierarchy.png) The "Any" Type

Every type, whether built-in or user-defined, has the any type at the root of its inheritance tree. By definition, the any type is a type which imposes no restrictions, i.e. any instance of data is valid against it.

#### Object Type

All types that have the built-in object base type in their inheritance tree MAY use the following facets in their type declarations:

| **Facet** | **Description** |
| --- | --- |
| properties? | The [properties](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#property-declarations) that instances of this type can or must have. |
| minProperties? | The minimum number of properties allowed for instances of this type. |
| maxProperties? | The maximum number of properties allowed for instances of this type. |
| additionalProperties? | A Boolean that indicates whether an object instance MAY contain [additional properties](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#additional-properties).  **Default:** true |
| discriminator? | Determines the concrete type of an individual object at runtime when, for example, payloads contain ambiguous types due to unions or inheritance. The value must match the name of one of the declared properties of a type. Unsupported practices are inline type declarations and [using discriminator](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#using-discriminator) with non-scalar properties. |
| discriminatorValue? | Identifies the declaring type. Requires including a discriminator facet in the type declaration. A valid value is an actual value that might identify the type of an individual object and is unique in the hierarchy of the type. Inline type declarations are not supported.  **Default:** The name of the type |

Example:

#%RAML 1.0

title: My API With Types

types:

Person:

type: object

properties:

name:

required: true

type: string

##### Property Declarations

Properties of object types are defined using the OPTIONAL **properties** facet. The RAML Specification calls the value of the properties facet a "properties declaration". The properties declaration MUST be a map of keys and values. The keys are valid property names for declaring a type instance. The values MUST be either a name of a type or an inline type declaration.

The properties declaration MAY specify whether a property is required or optional. Alternatively, a trailing question mark (?) in the key name MAY be used to indicate that a property is optional.

| **Facet** | **Description** |
| --- | --- |
| required? | Specifies that the property is required or not.  **Default:** true. |

The following example declares an object type having two properties:

types:

Person:

properties:

name:

required: true

type: string

age:

required: false

type: number

The following example shows a common idiom:

types:

Person:

properties:

name: string # equivalent to ->

# name:

# type: string

age?: number # optional property; equivalent to ->

# age:

# type: number

# required: false

When the required facet on a property is specified explicitly in a type declaration, any question mark in its property name MUST be treated as part of the property name rather than as an indicator that the property is optional.

For example, in

types:

profile:

properties:

preference?:

required: true

The profile type has a property named preference? that includes the trailing question mark. The following snippets show two ways of making preference? optional:

types:

profile:

properties:

preference?:

required: false

or

types:

profile:

properties:

preference??:

Note:

When an object type does not contain the "properties" facet, the object is assumed to be unconstrained and therefore capable of containing any properties of any type.

##### Additional Properties

By default, any instance of an object MAY have additional properties beyond those specified in its data type properties facet. Assume the following code is an instance of the data type Person that is described in the previous section.

Person:

name: "John"

age: 35

note: "US" # valid additional property `note`

The property note is not explicitly declared in the Person data type, but is valid because all additional properties are valid by default.

To restrict the addition of properties, you MAY set the value of the additionalProperties facet to false, or you MAY specify regular expression patterns that match sets of keys and restrict their values. The latter are called "pattern properties". The patterns are delineated by pairs of opening and closing / characters, as follows:

#%RAML 1.0

title: My API With Types

types:

Person:

properties:

name:

required: true

type: string

age:

required: false

type: number

/^note\d+$/: # restrict any properties whose keys start with "note"

# followed by a string of one or more digits

type: string

This pattern property restricts any additional properties whose keys start with "note" followed by a string of one or more digits. Consequently, the example of an object instance that declares an additional note1 property with the value "US" is valid, but the property note2 is invalid with a non-string value:

Person:

name: "John"

age: 35

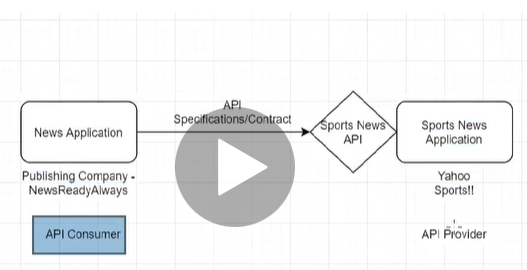
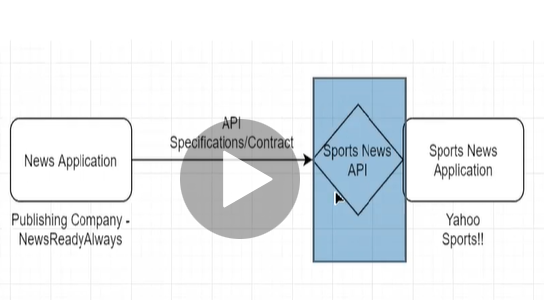
note1: "US" # valid

note2: 123 # not valid as it is not a string

note: 123 # valid as it does not match the pattern

Learn API design with RAML using Anypoint Designer , Mocking Service , Exchange & API Notebook

Introduction to API

API stands for Application Programming Interface which helps to provide a communication between two or more systems.eg: let assume newsreadyalways is a company which is publishing the news via news applications n for showing sports news it need to connect with yahoo Sports which hosts sports new application that provides sports news.In order to display the sports news from sports news application(Yahoo) to News application(NewsReadyAlways) there should be a mutual contract between newsreadyalways(API Consumer) and the Yahoo Sports(API Provider).The mutual contract means we have to create API Specification which is used for creating Sports news API,

USE Case-BookMyHotel

* Travel company wants to create BOOKMYHOTEL website
* This website is used for searching the hotel and booking the hotel.
* Search of hotel are done by city/State/Country
* MULEESB integration is used to connect the BookMyHotel website and different Hotels.

Traveller/Agents

Search and Book Hotels

BookMyHotel Website

Registers

Hotel 1

Hotel 2

Hotel N

Intro to Rest/RAML

There are different types of standards for designing API

* REST API
* SOAP API
* RPC API

REST API(Representational State Transfer)

* Uses HTTP as Application Protocol

-HTTP Methods like get,post,delete,put.patch are used to interact between API Provider and API Consumer so actions are mapped to HTTP Methods.Responses in HTTP status code like 200,201,400,405,401 and the body.

* Resource Based

-REST is resource based and not actions based unlike object in oop or entity in db.

-Resources are nouns.

Registration

API

Hotel

X

Bookings

getRegistration()

getHotel()

getBokings()

-Resources are 2 types

1. Collection Resources-Represents Collection like Registrations,Bookings,Hotels where Registrations is collections of all registrations,Bookings is collections of all bookings etc.
2. Nested Resource-Represents a specific Resource like booking,registration,Hotel where each booking represents a specific booking based on unique identifier like bookingID likewise for registration n hotel

* Representation

-Representation can be in json,xml etc n it is exchanged between provider n consumer for communication. The media type is used to specify representation .

* Uniform Interface

-Specify actions in terms of HTTP Methods

1. GET-Gets current status of resource
2. POST-Adds new resource
3. DELETE-Deletes a resource
4. PUT-Completely updates a resource if exists ,if not then Adds the resource
5. PATCH- Partially updates a resource

-Response as HTTP Status code n body

* Stateless

-No state is maintained

-States are maintained in client side.

RAML

-Stands for Restful Api Modeling Language

-RAML is built on Yaml

-RAML earlier version 0.8 n latest version is 1.0

-RAML 1.0 supports YAML 1.2

-YAML Eg:

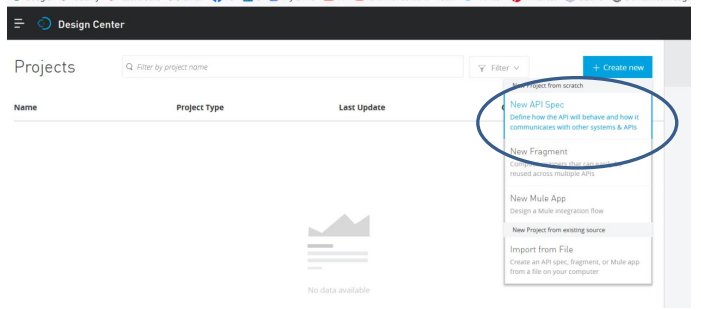
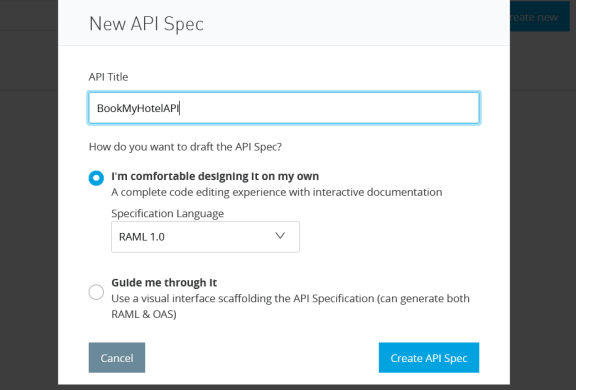
Dbname:surya

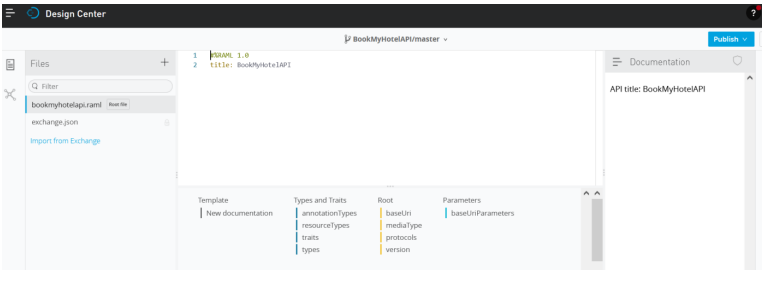
usernm:Kashinath

pwd:123

API Designer

-Mulesoft’s Anypoint Platform provides Design center in which we have API Designer(Used to build API Specification) and Flow Designer(Used to build Integration applications)

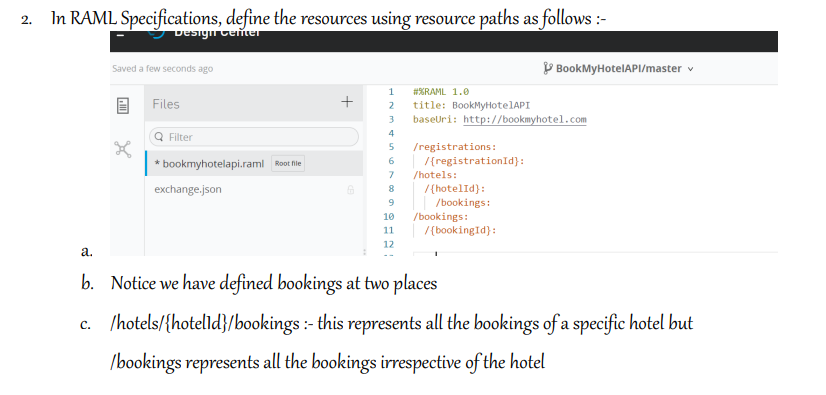
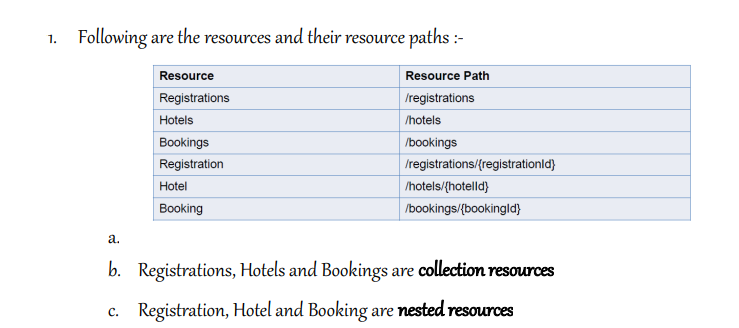
 

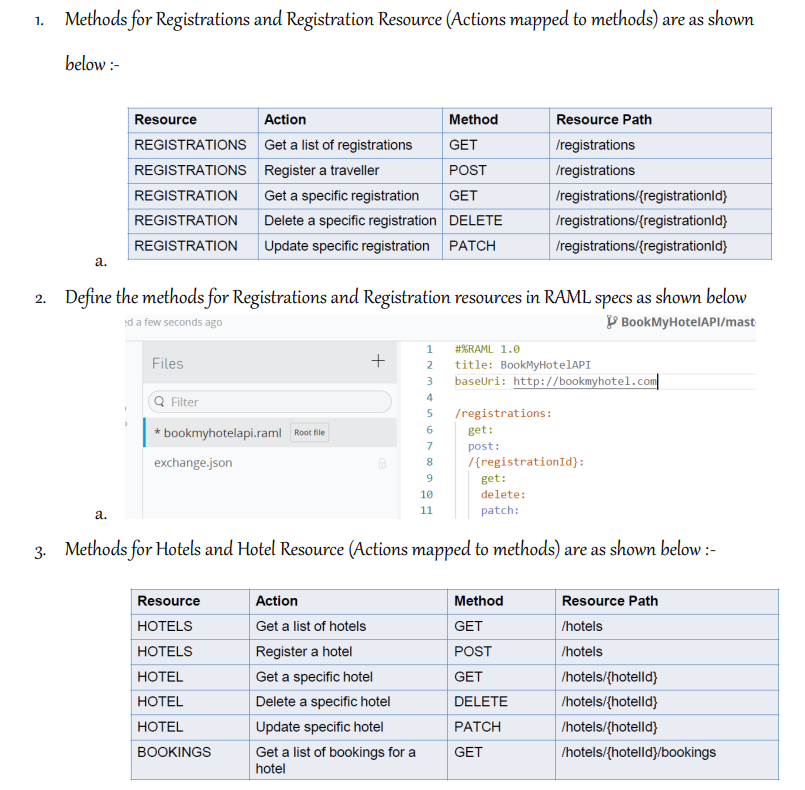
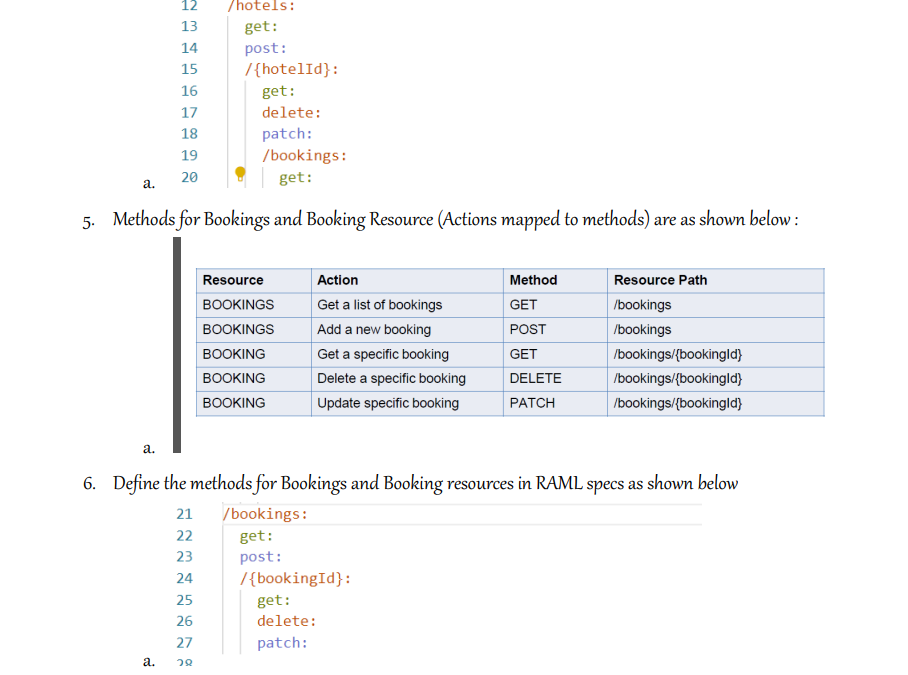
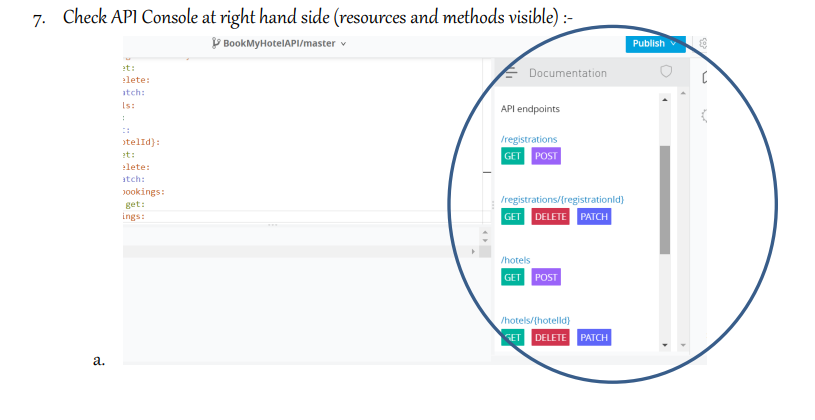


Resource

-Resources are identified by URIs.

-As we discussed and identified resources for our bookmyhotel use case that is /Registrations(for all registrations,/registration/{registrationid} for a specific registration likewise for bookings n hotels,hotel

  Methods:-

Media Type:

-Media type can be format of the data in which API consumer sends a request to API Provider or format of the data in which API Provider sends a response to API Consumer.

-Media type can be application/json ,application/xml or any other format.

-We have to set media type for request body as well as response body at that level otherwise what is specified at root level will be used for applying media-type.

Response Body

Media Type

Request Body

application/json

application/xml

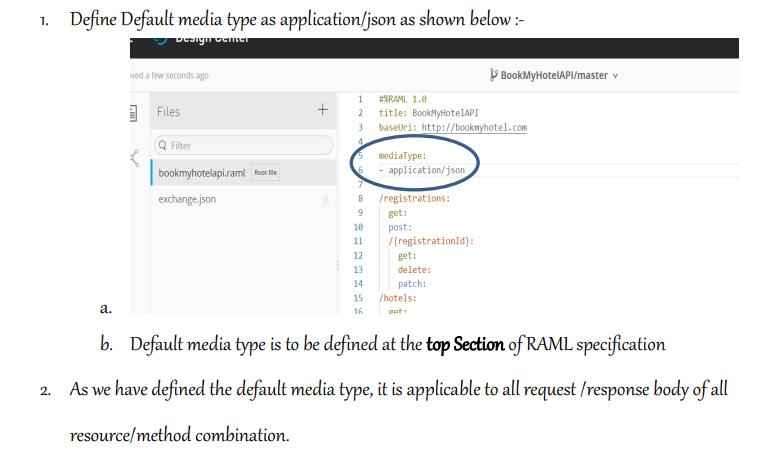
-Media Type specify Headers

1. Content-Type:application/json

-when API consumer sends a request from his side to the API provider with Content-Type:application/json, API provider understands that the request is in application/json

1. Accept: application/json

- when API consumer sends a request from his side to the API provider with Accept: application/json, API provider understands that the response to be sent to API Conumer should be in application/json format.



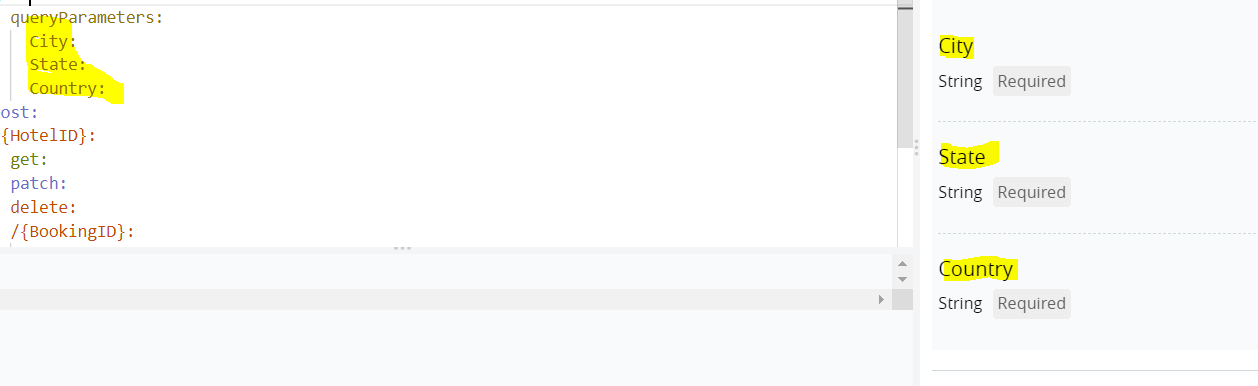
Query Parameters:

-Query Parameters are used to sort or filter the particular resource.

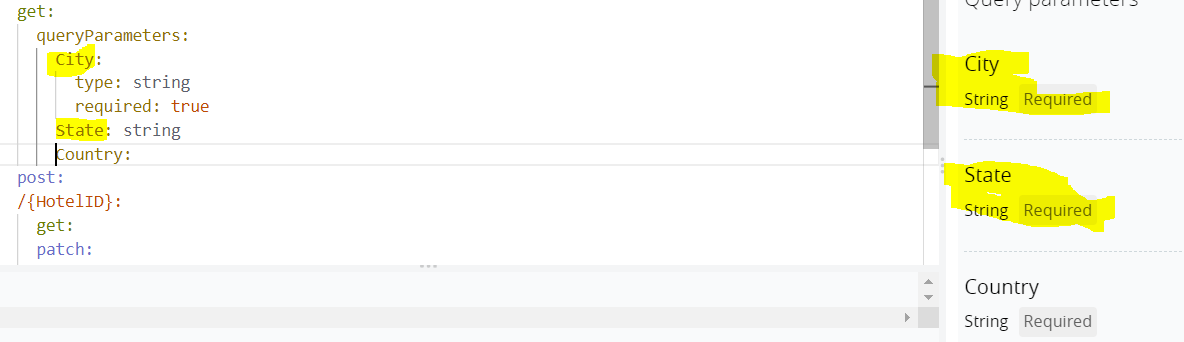
-To specify query parameters for a particular resource’s method(as specified in UseCase for /Hotels get method,we have to Search for hotel are done by city/State/Country.

-So our service url will become like

<http://www.bookmyhotel.com/Hotels?City=value1&State=value2&Country=value3>

-If we specify any query parameters like below by default datatype=string and required=true (query parameter is not optional) 

Or else same we can write in code as,

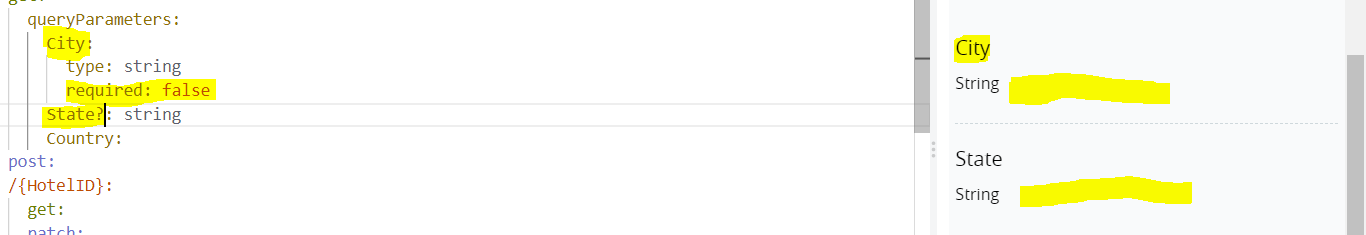


We can specify the datatype and Reqired query parameter condition with the help of type facet (as I specified in City query Parameter) or inline as I specified in above pic for State query Parameter.

-To make any query parameter as optional we can do that by 2 ways

1. by specifying queryparameter as Required: false

2. by specifying queryparameter as queryparametername?



-Default is used to set the default value if no value is given then default value is considered.

-minlength n maxlength: specifies minlength n maxlength of the value.

-pattern is used for defining the pattern of value .

-description is used to give description of request

-Displayname is used to give alternate name

-enum is used as dropdown with values specified for the given

