|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr.No** | **Component** | **Description** | **Usage** | **Examples** | **Best Practice for this component** |
| **1** | **Resources** | A resource defined at the root level is called a top-level resource. the final URL to the resource is based on the information provided in the document root section. | This is the most fundamental element in RAML since the entire API implementation is around resources. Any methods and parameters nested under these top level resources belong to and act upon that resource. | #%RAML 0.8  title: GitHub API  version: v3  baseUri: https://api.github.com  /gists:  displayName: Gists | * The resource defines the entity which houses the information to be retrieved, added, updated or deleted. * Resources have to be preceded by a forward slash (/) and end with a colon (:).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). |
| **2** | **baseUri** | The **baseUri** is the URL where the API will be hosted.  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. After the API is implemented  and can be accessed at a service endpoint, the API definition must contain a baseUri property.  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) | **BaseUri** 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.  The use of the baseUri field is optional to allow describing APIs that have not yet been implemented. | #%RAML 0.8  title: Salesforce Chatter REST API  version: v28.0  baseUri: https://na1.salesforce.com/services/data/{version}/chatter | * The baseUriParameters node has the same structure and semantics 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 value of the baseUri node is a string that MUST conform to the URI specification [RFC2396](https://www.ietf.org/rfc/rfc2396.txt) or a [Template URI](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#template-uri). |
| **3** | **HTTP Methods** | RESTful API methods are operations that are performed on a resource. | Use of HTTP methods are:   * **GET**—Retrieve the information defined in the request URI. * **PUT**—Replace the addressed collection. At the object-level, create or update it. * **POST**—Create a new entry in the collection. This method is generally not used at the object-level. * **DELETE**—Delete the information defined in the request URI. | #%RAML 0.8  title: ZEncoder API  version: v2  baseUri: https://app.zencoder.com/api/{version}  /jobs:  post:  description: Create a Job  get:  description: List Jobs | An alternate, human-friendly method name in the context of the resource.   * If the display Name node is not defined for a method, documentation tools SHOULD refer to the resource by its key, which acts as the method name. * 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). * Detailed information about any query parameters needed by this method. Mutually exclusive with queryString. * Detailed information about any request headers needed by this method. |
| **4** | **Title** | The title property is a short plain text description of the RESTful API. | Title component in RAML is used to give a name of API. | #%RAML 1.0  title: My API | Title of the document. Its value MUST be a non-empty string. |
| **5** | Nested Resources | A resource which is defined as a child of another resource is called a nested resource. | A nested resource can itself have a child (nested) resource, creating a multiply-nested resource. | #%RAML 0.8  title: GitHub API  version: v3  baseUri: https://api.github.com  /gists:  displayName: Gists  /public:  displayName: Public Gists | A nested resource, which is identified as any node whose name begins with a slash (/), and is therefore treated as a relative URI. |
| **6** | ****Responses**** | The resources and methods sections of this document describe HTTP requests.  This section describes the HTTP responses to method invocations on resources. | * The **responses** node of a method on a resource describes the possible responses to invoking that method on that resource. * The value of **responses** is a map where each key name represents that a possible HTTP status codes for that method on that resource. * The values describe the corresponding responses. | responses:      200:        body:          application/json:            example: |              {                "data":{                   "id": "postID",                   "title": "My Post",                   },                 "success": true,                 "status": 200               }           application/xml: | * Keys are often numeric, for example 200 or 204. Processors MUST treat these numeric keys as string keys in all situations. * For example, '200' and 200 MUST be treated as duplicate keys, and therefore, are not allowed simultaneously. |
| **7** | URI Parameters | The braces { } around property names define URI parameters.  They represent placeholders in each URI and do not reference root-level RAML file. The added lines represent the resources. | URI parameters can be used to define a relative URI of a resource that contains variable elements. | /posts:    /{id}:    /{title}:    /{content}:  /{post}:     /{id}:    /{author}:     /{id}:     /{authorName}  /authors:  /{id}:  /{authorName}:      /readers:  /{id}:  /{readername}:  /{commentid}: | * URI parameters are denoted by surrounding curly braces {} in RAML; for example, {title} for posts, {authorName} for authors, etc. * To access an individual post, say post with id=1, the URL is http://myblog.com/v1/posts/1. |
| **8** | Query Parameters | * Query parameters are a mixture of meta information like description, access\_tokens (required by server), whether required or not. * Query parameters may be defined on any number of nested resources and their methods. | The **queryParameters** node specifies the set of query parameters from which the query string is composed. When applying the restrictions defined by the API, processors MUST regard the query string as a set of query parameters according to the URL encoding format. | /foos:    get:      description: List all Foos matching query criteria      queryParameters:        name?: string        ownerName?: string | * The query string in a URL is defined in [RFC3986](https://www.ietf.org/rfc/rfc3986.txt) as the part of the URL following the question mark separator (?) and preceding any fragment (#) separator. * The query string can be specified either by the **queryString** node or by the **queryParameters**node. * The queryString and queryParameters nodes are mutually exclusive: processors MUST NOT allow both to be specified, explicitly or implicitly, on the same method of the same resource. |
| **9** | Protocols | If the protocols node is not explicitly specified, one or more protocols included in the baseUri node is used; if the protocols node is explicitly specified, the node specification overrides any protocol included in the baseUri node. | The **protocols** node specifies the protocols that an API supports. | #%RAML 1.0  title: Salesforce Chatter REST API  version: v28.0  protocols: [ HTTP, HTTPS ]  baseUri: https://na1.salesforce.com/services/data/{version}/chatter | The protocols node MUST be a non-empty array of strings, of values HTTP and/or HTTPS, and is case-insensitive. |
| **10** | **Body** | Some method verbs expect the resource to be sent as a request body.  For example, to create a resource, the request must include the details of the resource to create. | Resources CAN have alternate representations. For example, an API might support both JSON and XML representations. | /jobs:  post:  description: Create a Job  body:  text/xml: !!null  application/json: !!null | A method's body is defined in the body property as a hashmap, in which the key MUST be a valid media type. |
| **11** | API Version | If the RAML API definition is targeted to a specific API version, the API definition MUST containa version  property. | * The version property is OPTIONAL and should not be used if:The API itself is not versioned. * The API definition does not change between versions. The API architect can decide whether a change to user documentation elements, but no change to the API's resources, constitutes a version change. | #%RAML 1.0  title: e-BookMobile API  baseUri: http://api.e-bookmobile.com/{version}  version: v1 | * The API architect MAY use any versioning scheme so long as version numbers retain the same format. * For example, "v3", "v3.0", and "V3" are all allowed, but are not considered to be equal. |
| **12** | **Security Schemes** | To apply a security Scheme definition to every method in an API, the API MAY be defined using the secured  By attribute.  This specifies that all methods in the API are protected using that security scheme.  An optional **security**  **Schemes** node is defined  for the RAML document root.  Each authentication pattern supported by the API must be expressed as a component of the **securitySchemes**node value. | * A securityScheme may also be applied to a resource by using the securedBy key, which is equivalent to applying the securityScheme to all methods that may be declared, explicitly or implicitly, by defining the resourceTypes or traits property for that resource. * To indicate that the method may be called without applying any securityScheme, the method may be annotated with the nullsecurityScheme. | #%RAML 0.8  title: Dropbox API  version: 1  baseUri: https://api.dropbox.com/{version}  securedBy: [oauth\_2\_0]  securitySchemes:  1. oauth\_2\_0: !include oauth\_2\_0.yml  1. oauth\_1\_0: !include oauth\_1\_0.yml  /users:  get:  securedBy: [oauth\_2\_0, oauth\_1\_0] | The value of securitySchemes is a map having key-value pairs that map security scheme names to security scheme declarations.   * One API-supported authentication method is allowed. The value MUST be one of the following methods: OAuth 1.0, OAuth 2.0, Basic Authentication, Digest Authentication, Pass Through, x-<other>. * Information that MAY be used to describe a security scheme. 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). * The [settings](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#settings) attribute MAY be used to provide security scheme-specific information. |
| **13** | Media Types | Specifying the  **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. | 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.  The media type applies to requests having a body, the expected responses, and examples using the same sequence of media type strings. | #%RAML 1.0  title: New API  mediaType: [ application/json, application/xml ] | The value of the mediaType node MUST be a sequence of media type strings or a single media type string.  Each value needs to conform to the media type specification in [RFC6838](https://tools.ietf.org/html/rfc6838). |
| **14** | **Data Types** | RAML 1.0 introduces the notion of **data types**, which provide a concise and powerful way of describing the data in an API. Data types add rules for validating data against a type declaration.  Valid data adheres to all rules for the type. | 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.  Extending types MUST NOT create any cyclic dependencies. A type can be extended inline. | #%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 | RAML Types in a nutshell:   * Types are similar to Java classes. * You can define types that inherit from other types. * Types are split into four families: external, object, array, and scalar. * Types can define two types of members: properties and facets. * Only object types can declare properties. All types can declare facets. * To specialize a scalar type, you implement facets, giving already defined facets a concrete value. * To specialize an object type, you define properties. |
| **15** | **Modularization** | RAML provides several mechanisms to help modularize the ecosystem of an API specification:   * Includes * Libraries  Includes RAML processors MUST support the **!include** tag, which specifies the inclusion of external files into the API specification. Being a YAML tag, the exclamation point (!) prefix is required.  Libraries RAML libraries are used to combine any collection of data type declarations, resource type declarations, trait declarations, and security scheme declarations into modular, externalized, reusable groups. | Includes In an API specification, the !include tag is located only in a node value position. The !include tag MUST be the value of a node, which assigns the contents of the file named by the !include tag to the value of the node. Librarieslibraries are intended to define common declarations in external documents, which are then included where needed, libraries can also be defined inline. | Includes #%RAML 1.0  title: My API with Types  types: !include myTypes.raml Libraries #%RAML 1.0 Library  usage: |  Use to define some basic file-related constructs.  types:  File:  properties:  name:  length:  type: integer  traits:  drm:  headers:  drm-key:  resourceTypes:  file:  get:  is: [ drm ]  put:  is: [ drm ] | Includes  * A path that begins with a single slash (/) and is interpreted relative to the root RAML file location. * A path that neither begins with a single slash (/) nor constitutes a URL, and is interpreted relative to the location of the included file.  LibrariesThe definition of each node is the same as that of the corresponding node at the root of a RAML document. A library supports annotation node like any other RAML document.Describes the content or purpose of a specific library. The 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). |
| **16** | Traits | * Traits can be declared using the **traits** node at the root of the API definition. * The value of this node is a map where key names become names of traits that can be referenced throughout the API, and values are trait declarations. | A trait may also be applied to a resource by using the is key, which is equivalent to applying the trait to all methods for that resource, whether declared explicitly in the resource definition or inherited from a resource type. | #%RAML 1.0  title: Example API  version: v1  resourceTypes:  collection: !include resourceTypes/collection.raml  member: !include resourceTypes/member.raml  traits:  secured: !include traits/secured.raml  rateLimited: !include traits/rate-limited.raml | * The  **usage** of trait provides instructions about how and when the trait SHOULD be used. Documentation generators MUST describe this node in terms of the characteristics of the resource and method, respectively. * However, the resources and methods MUST NOT inherit the usage node. Neither resources nor methods allow a node named usage. |