#RAML###RESTful###API

RAML###RESTful API Modeling Language#RESTful API#######MuleSoft#########

RAML#######API########2017#3#13#######1.0####YAML 1.2################RAML####API############RESTful API############HTML###########SDK################

1.

RAML##########API###################

RAML API#################RAML###

##GitHub v3##API###RAML 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 ]
/search:
  /code:
    type: collection
    get:
```

#####RAML############

###	##
title	####API#####

description?	######API########## markdown[1] # ##
version?	##API#####"v1"#######
baseUri?	######## #URIs[2]#### ##URI[3].
baseUriParameters?	#Uri[4]########## #
protocols?	#API### ##[5]#
mediaType?	######### ######[6]##"application/json"#
documentation?	API### ##[7]#
schemas?	#RAML 0.8####"types"########## ########"types"####XML#JSON# schemas#
types?	(##)##[8]###
traits?	traits[9]####
resourceTypes?	API### ####[10] #
annotationTypes?	##### ####[11]###

[1] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#markdown
[2] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#base-uri-and-base-uri-parameters
[3] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#template-uri
[4] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#base-uri-and-base-uri-parameters
[5] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#protocols
[6] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#default-media-types
[7] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#user-documentation
[8] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#defining-types
[9] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#resource-types-and-traits
[10] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#resource-types-and-traits
[11] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#declaring-annotation-types

(<annotationName>)?	API### ##[12]##################### ################
securitySchemes?	############ ##schemes[13]#
securedBy?	##schemes[14]#
uses?	#### #[15] .
/<relativeUri>?	#######URIs#######API### ### #[16]############################# # /users # /{groupId}#

"schemas" # "types" ##"types"### #"schemas"############RAML#####"schemas"###

1.1. ####[17]

###**documentation**###################API########################API# #################

documentation##########documents###document##################

##	##
title	###############
content	####################### markdown[18]###

#####

[12] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#annotations
[13] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#security-schemes
[14] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#applying-security-schemes
[15] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#libraries
[16] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#resources-and-nested-resources
[17] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#user-documentation
[18] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#markdown

```
#%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
```

1.2. Base URI # Base URI##[19]

### **baseUri** ####API##URI################### RFC2396[20] ## URI##[21] ###

## baseUri #### URI##[22] #######base URI###

URI Parameter	#
version	#######

#####baseUri##URI######### **baseUriParameters** ###API######### ##baseUriParameters##### uriParameters[23] ##################URI#####

### RAML API ##### URI##[24] ###URI#

```
#%RAML 1.0
title: Salesforce Chatter REST API
```

[19] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#base-uri-and-base-uri-parameters
[20] https://www.ietf.org/rfc/rfc2396.txt
[21] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#template-uri
[22] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#template-uri
[23] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#template-uris-and-uri-parameters
[24] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#template-uri

```
version: v28.0
baseUri: https://na1.salesforce.com/services/data/{version}/chatter
```

############ base URI ###

```
#%RAML 1.0
title: Amazon S3 REST API
version: 1
baseUri: https://{bucketName}.s3.amazonaws.com
baseUriParameters:
  bucketName:
    description: The name of the bucket
```

#baseAPI##########/)################################## `http://api.test.com/common/users` # `http://api.test.com/common/users/groups`#

```
baseUri: http://api.test.com/common/
/users:
  /groups:
```

######################`//api.test.com//common/`, `//api.test.com//common//users/`, and `//api.test.com//common//users//groups//`.

```
baseUri: //api.test.com//common//
/:
  /users/:
    /groups//:
```

1.3. ##[25]

### **protocols** #####API######## protocaols #############protocols##### baseUri####protocals#################HTTP#/#HTTPS########

########

```
#%RAML 1.0
title: Salesforce Chatter REST API
```

[25] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#protocols

```
version: v28.0
protocols: [ HTTP, HTTPS ]
baseUri: https://na1.salesforce.com/services/data/{version}/chatter
```

1.4. ######[26]

mediaType##########################

mediaType###################URL########## RFC6838[27] ############ ######

#######json######RAML##################################API####### JSON######

```
#%RAML 1.0
title: New API
mediaType: application/json
```

####################Json#xml#RAML###

```
#%RAML 1.0
title: New API
mediaType: [ application/json, application/xml ]
```

###############Json#xml#########POST#GET############ /list ##### JSON#XML#####/send######JSON########## body[28]#

```
#%RAML 1.0
title: New API
mediaType: [ application/json, application/xml ]
types:
  Person:
  Another:
/list:
  get:
    responses:
```

[26] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#default-media-types
[27] https://tools.ietf.org/html/rfc6838
[28] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#bodies

```
      200:
        body: Person[]
/send:
  post:
    body:
      application/json:
        type: Another
```

1.5. #######[29]

securedBy###################schemes####API#####################
######security scheme#name##### Applying Security Schemes[30]##########
############security schemes#

##########API######OAuth 2.0##OAuth 1.1#######

```
#%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
```

2. RAML

2.1. ##[31]

RAML 1.0###**####**##################API##############################
########

########URI##
#######API##
#####################

[29] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#default-security
[30] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#applying-security-schemes
[31] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#introduction-1

#######RAML#########User########firstname, lastname, ## age ######## ##########string#number#########payload######User###schema##

```
#%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#######JSON schema#######RAML########JSON#XML schemas### #######RAML##################JSON#XML#schemas################# ########################

```
#%RAML 1.0
title: My API with Types
mediaType: application/json
types:
  Org:
    type: object
    properties:
      onCall: AlertableAdmin
      Head: Manager
  Person:
    type: object
    properties:
      firstname: string
      lastname:  string
      title?:    string
  Phone:
    type: string
    pattern: "[0-9|-]+"
  Manager:
    type: Person
    properties:
      reports: Person[]
      phone:  Phone
  Admin:
    type: Person
    properties:
      clearanceLevel:
        enum: [ low, high ]
  AlertableAdmin:
    type: Admin
    properties:
      phone: Phone
  Alertable: Manager | AlertableAdmin
/orgs/{orgId}:
  get:
    responses:
      200:
        body:
          application/json:
            type: Org
```

2.2.

#########

RAML#########Java#####XSD#Json Schemas###

RAML#####

- Types#Java#####
  - Types###JSON Schema#XSD##################
- #####################
  - #Java###RAML##########
- Types##############################scalar#######
- Types#########**properties####**#**facets###**##########
  - **Properties####**#############
  - **Facets###**##############facet############minLength#######maxLength(######

- ##########################facets####
- #######facets##facets###########scalar###
- ####################

2.3. ####[32]

########API################API######**types**##################RAML ####### ####map#[33] ###################

```
types:
  Person: # key name
    # value is a type declaration
```

2.4. ####[34]

#############facets###########facets############################## ########**#####**#####################facet#

Facet	##
default?	#######API##################### ###########API#######default### ############API################ #################default####### ##URI#################facet#URI ############################
schema?	###"type"####RAML 0.8######## #####RAML API######Facet#### #"type"#####"type"####XML#Json#
type?	#######################a) ##### #### b) RAML#####(object##, array# #, ##scalar### c) ##############

[32] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#defining-types
[33] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#type-declarations
[34] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#type-declarations

example?	###"examples"facet######## facet######## Examples[35]#
examples?	############### Examples[36]#
displayName?	###facet#################
description?	###################### markdown[37]###
(<annotationName>)?	#API####https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#annotations[##]###################
facets?	#####Map################## facets####### #####Facets[38] #
xml?	###### #####XML###[39]###
enum?	#####################facet###### ######facet###########

"schema"#"type"###facets###################

```
types:
  Person:
    schema: # invalid as mutually exclusive with `type`
    type: # invalid as mutually exclusive with `schema`
```

```
/resource:
  get:
    responses:
```

[35] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#defining-examples-in-raml
[36] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#defining-examples-in-raml
[37] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#markdown
[38] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#user-defined-facets
[39] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#xml-serialization-of-type-instances

```
      200:
        body:
          application/json: # start type declaration
            schema: # invalid as mutually exclusive with `type`
            type: # invalid as mutually exclusive with `schema`
```

#####"type"###"schema"###schema###RAML###########"type"######
XML#JSON schema#

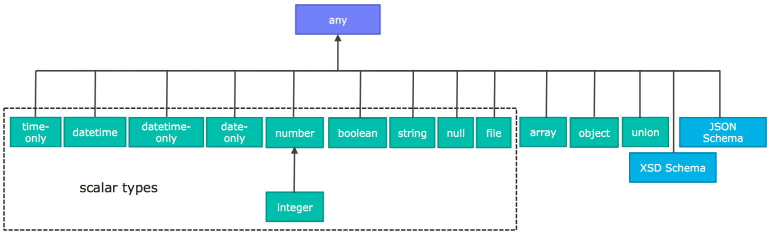
2.5.

RAML##############

- any##[40]
- object##[41]
- array##[42]
- union##[43]#####
- scalar##[44]#number##, boolean##, string###, date-only###, time-only###, datetime-only#####, datetime####, file##, integer##, ##nil##

#########RAML######### JSON#XML schema[45]#

####################### `any` ######



[40] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#the-any-type
[41] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#object-type
[42] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#array-type
[43] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#union-type
[44] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#scalar-types
[45] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#using-xml-and-json-schema

"Any"

######## any###any#any########### RAML##any######Java####Object#########Java############ Object##

any####facets#

Object####[46]

############################facets#

Facet	##
properties?	############# ##[47] #
minProperties?	##################
maxProperties?	##################
additionalProperties?	######### #####[48] # **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`[49] with non-scalar properties.

46 https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#object-type
47 https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#property-declarations
48 https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#additional-properties
49 https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#using-discriminator

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

An object type is created by explicit inheritance from the built-in type object:

```
#%RAML 1.0
title: My API With Types
types:
  Person:
    type: object
    properties:
      name:
        required: true
        type: string
```

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[50]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

[50] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#property-declarations

declaration". The properties declaration is a map of keys and values. The keys are valid property names for declaring a type instance. The values are either a name of a type or an inline type declaration.

The properties declaration can specify whether a property is required or 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 is 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.

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[51] Additional Properties

By default, any instance of an object can 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 can set the value of the `additionalProperties` facet to `false`, or you can 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:
```

[51] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#additional-properties

```
        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 `note` property with the value "US" is valid, but the same property is invalid with a non-string value:

```
Person:
  name: "John"
  age: 35
  note: 123 # not valid as it is not a string
  address: "US" # valid as it does not match the pattern
```

To force all additional properties to be strings, regardless of their keys, use:

```
#%RAML 1.0
title: My API With Types
types:
  Person:
    properties:
      name:
        required: true
        type: string
      age:
        required: false
        type: number
      //: # force all additional properties to be a string
        type: string
```

If a pattern property regular expression also matches an explicitly declared property, the explicitly declared property definition prevails. If two or more pattern property regular expressions match a property name in an instance of the data type, the first one prevails.

Moreover, if `additionalProperties` is `false` (explicitly or by inheritance) in a given type definition, then explicitly setting pattern properties in that definition is not allowed. If `additionalProperties` is `true` (or omitted) in a given type definition, then pattern properties are allowed and further restrict the additional properties allowed in that type.

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[52] Object Type Specialization

You can declare object types that inherit from other object types. A sub-type inherits all the properties of its parent type. In the following example, the type `Employee` inherits all properties of its parent type `Person`.

```
#%RAML 1.0
title: My API With Types
types:
  Person:
    type: object
    properties:
      name:
        type: string
  Employee:
    type: Person
    properties:
      id:
        type: string
```

[52] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#object-type-specialization

A sub-type can override properties of its parent type with the following restrictions: 1) a required property in the parent type cannot be changed to optional in the sub-type, and 2) the type declaration of a defined property in the parent type can only be changed to a narrower type (a specialization of the parent type) in the sub-type.

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[53]Using Discriminator

When payloads contain ambiguous types due to unions or inheritance, it is often impossible to discriminate the concrete type of an individual object at runtime. For example, when deserializing the payload into a statically typed language, this problem can occur.

A RAML processor might provide an implementation that automatically selects a concrete type from a set of possible types, but a simpler alternative is to store some unique value associated with the type inside the object.

You set the name of an object property using the `discriminator` facet. The name of the object property is used to discriminate the concrete type. The `discriminatorValue` stores the actual value that might identify the type of an individual object. By default, the value of `discriminatorValue` is the name of the type.

Here's an example that illustrates how to use `discriminator`:

```
#%RAML 1.0
title: My API With Types
types:
```

[53] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#using-discriminator

```
  Person:
    type: object
    discriminator: kind # refers to the `kind` property of object
 `Person`
    properties:
      kind: string # contains name of the kind of a `Person` instance
      name: string
  Employee: # kind can equal `Employee`; default value for
 `discriminatorValue`
    type: Person
    properties:
      employeeId: integer
  User: # kind can equal `User`; default value for `discriminatorValue`
    type: Person
    properties:
      userId: integer
```

```
data:
  - name: A User
    userId: 111
    kind: User
  - name: An Employee
    employeeId: 222
    kind: Employee
```

You can also override the default for `discriminatorValue` for each individual concrete class. The following example replaces the default value of `discriminatorValue` in initial caps with lowercase:

```
#%RAML 1.0
title: My API With Types
types:
  Person:
    type: object
    discriminator: kind
    properties:
      name: string
      kind: string
  Employee:
    type: Person
    discriminatorValue: employee # override default
    properties:
      employeeId: string
  User:
    type: Person
    discriminatorValue: user # override default
    properties:
      userId: string
```

```
data:
  - name: A User
    userId: 111
    kind: user
  - name: An Employee
    employeeId: 222
    kind: employee
```

Neither `discriminator` nor `discriminatorValue` can be defined for any inline type declarations or union types.

```
# valid whenever there is a key name that can identify a type
types:
  Device:
    discriminator: kind
    properties:
      kind: string
```

```
# invalid in any inline type declaration
application/json:
   discriminator: kind
   properties:
     kind: string
```

```
# invalid for union types
PersonOrDog:
   type: Person | Dog
   discriminator: hasTail
```

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[54] Array Type

Array types are declared by using either the array qualifier `[]` at the end of a type expression[55] or `array` as the value of a `type` facet. If you are defining a top-level array type, such as the `Emails` in the examples below, you can declare the following facets in addition to those previously described to further restrict the behavior of the array type.

Facet	Description
uniqueItems?	Boolean value that indicates if items in the array MUST be unique.
items?	Indicates the type all items in the array are inherited from. Can be a reference to an existing type or an inline type declaration[56].
minItems?	Minimum amount of items in array. Value MUST be equal to or greater than 0. **Default:** `0`.

[54] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#array-type
[55] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#type-expressions
[56] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#type-declaration

maxItems?	Maximum amount of items in array. Value MUST be equal to or greater than 0. **Default:** `2147483647`.

Both of the following examples are valid:

```
types:
  Email:
    type: object
    properties:
      subject: string
      body: string
  Emails:
    type: Email[]
    minItems: 1
    uniqueItems: true
    example: # example that contains array
      - # start item 1
        subject: My Email 1
        body: This is the text for email 1.
      - # start item 2
        subject: My Email 2
        body: This is the text for email 2.
```

```
types:
  Email:
    type: object
    properties:
      name:
        type: string
  Emails:
    type: array
    items: Email
    minItems: 1
    uniqueItems: true
```

Using `Email[]` is equivalent to using `type: array`. The `items` facet defines the `Email` type as the one each array item inherits from.

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[57] Scalar Types

RAML defines a set of built-in scalar types, each of which has a predefined set of restrictions.

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[58] String

A JSON string with the following additional facets:

Facet	Description
pattern?	Regular expression that this string should match.
minLength?	Minimum length of the string. Value MUST be equal to or greater than 0.

[57] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#scalar-types
[58] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#string

	Default: `0`
maxLength?	Maximum length of the string. Value MUST be equal to or greater than 0. **Default:** `2147483647`

Example:

```
types:
  Email:
    type: string
    minLength: 2
    maxLength: 6
    pattern: ^note\d+$
```

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[59]Number

Any JSON number including integer[60] with the following additional facets:

Facet	Description
minimum?	The minimum value of the parameter. Applicable only to parameters of type number or integer.
maximum?	The maximum value of the parameter. Applicable only to parameters of type number or integer.

59 https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#number
60 https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#integer

format?	The format of the value. The value MUST be one of the following: int32, int64, int, long, float, double, int16, int8
multipleOf?	A numeric instance is valid against "multipleOf" if the result of dividing the instance by this keyword’s value is an integer.

Example:

```
types:
  weight:
    type: number
    minimum: 3
    maximum: 5
    format: int64
    multipleOf: 4
```

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[61] Integer

A subset of JSON numbers that are positive and negative multiples of 1. The integer type inherits its facets from the number type[62].

```
types:
```

[61] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#integer
[62] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#number

```
  Age:
    type: integer
    minimum: 3
    maximum: 5
    format: int8
    multipleOf: 1
```

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[63] Boolean

A JSON boolean without any additional facets.

```
types:
  IsMarried:
    type: boolean
```

[63] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#boolean

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[64] Date

The following date type representations MUST be supported:

Type	Description
date-only	The "full-date" notation of RFC3339[65], namely `yyyy-mm-dd`. Does not support time or time zone-offset notation.
time-only	The "partial-time" notation of RFC3339[66], namely hh:mm:ss[.ff...]. Does not support date or time zone-offset notation.
datetime-only	Combined date-only and time-only with a separator of "T", namely yyyy-mm-ddThh:mm:ss[.ff...]. Does not support a time zone offset.
datetime	A timestamp in one of the following formats: if the *format* is omitted or set to `rfc3339`, uses the "date-time" notation of RFC3339[67]; if *format*

[64] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#date
[65] http://xml2rfc.ietf.org/public/rfc/html/rfc3339.html#anchor14
[66] http://xml2rfc.ietf.org/public/rfc/html/rfc3339.html#anchor14
[67] http://xml2rfc.ietf.org/public/rfc/html/rfc3339.html#anchor14

	is set to `rfc2616`, uses the format defined in RFC2616[68].

The additional facet *format* MUST be available only when the type equals *datetime*, and the value MUST be either `rfc3339` or `rfc2616`. Any other values are invalid.

```
types:
  birthday:
    type: date-only # no implications about time or offset
    example: 2015-05-23
  lunchtime:
    type: time-only # no implications about date or offset
    example: 12:30:00
  fireworks:
    type: datetime-only # no implications about offset
    example: 2015-07-04T21:00:00
  created:
    type: datetime
    example: 2016-02-28T16:41:41.090Z
    format: rfc3339 # the default, so no need to specify
  If-Modified-Since:
    type: datetime
    example: Sun, 28 Feb 2016 16:41:41 GMT
    format: rfc2616 # this time it's required, otherwise, the example
 format is invalid
```

[68] https://www.ietf.org/rfc/rfc2616.txt

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[69] File

The **file** type can constrain the content to send through forms. When this type is used in the context of web forms it SHOULD be represented as a valid file upload in JSON format. File content SHOULD be a base64-encoded string.

Facet	Description
fileTypes?	A list of valid content-type strings for the file. The file type / MUST be a valid value.
minLength?	Specifies the minimum number of bytes for a parameter value. The value MUST be equal to or greater than 0. **Default:** `0`
maxLength?	Specifies the maximum number of bytes for a parameter value. The value MUST be equal to or greater than 0. **Default:** `2147483647`

```
types:
  userPicture:
    type: file
```

[69] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#file

```
    fileTypes: ['image/jpeg', 'image/png']
    maxLength: 307200
  customFile:
    type: file
    fileTypes: ['*/*'] # any file type allowed
    maxLength: 1048576
```

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[70]Nil Type

In RAML, the type `nil` is a scalar type that allows only nil data values. Specifically, in YAML it allows only YAML's `null` (or its equivalent representations, such as `~`), in JSON it allows only JSON's `null`, and in XML it allows only XML's `xsi:nil`. In headers, URI parameters, and query parameters, the `nil` type only allows the string value "nil" (case-sensitive); and in turn an instance having the string value "nil" (case-sensitive), when described with the `nil` type, deserializes to a nil value.

In the following example, the type of an object and has two required properties, `name` and `comment`, both defaulting to type `string`. In `example`, `name` is assigned a string value, but comment is nil and this is *not* allowed because RAML expects a string.

```
types:
  NilValue:
    type: object
    properties:
      name:
      comment:
    example:
```

[70] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#nil-type

```
      name: Fred
      comment: # Providing no value here is not allowed.
```

The following example shows the assignment of the `nil` type to `comment`:

```
types:
  NilValue:
    type: object
    properties:
      name:
      comment: nil
    example:
      name: Fred
      comment: # Providing a value here is not allowed.
```

The following example shows how to represent nilable properties using a union:

```
types:
  NilValue:
    type: object
    properties:
      name:
      comment: nil | string # equivalent to ->
                           # comment: string?
    example:
      name: Fred
      comment: # Providing a value or not providing a value here is
 allowed.
```

Declaring the type of a property to be `nil` represents the lack of a value in a type instance. In a RAML context that requires *values* of type `nil` (vs just type declarations), the usual YAML `null` is used, e.g. when the type is `nil | number` you may use `enum: [ 1, 2, ~ ]` or more explicitly/verbosely `enum: [ 1, 2, !! null "" ]`; in non-inline notation you can just omit the value completely, of course.

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[71] Union Type

A union type is used to allow instances of data to be described by any of several types. A union type is declared via a type expression that combines 2 or more types delimited by pipe (`|`) symbols; these combined types are referred to as the union type's super types. In the following example, instances of the `Device` type may be described by either the `Phone` type or the `Notebook` type:

```
#%RAML 1.0
title: My API With Types
types:
  Phone:
    type: object
    properties:
      manufacturer:
        type: string
      numberOfSIMCards:
        type: number
      kind: string
  Notebook:
    type: object
    properties:
      manufacturer:
        type: string
      numberOfUSBPorts:
        type: number
      kind: string
  Device:
    type: Phone | Notebook
```

[71] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#union-type

An instance of a union type is valid if and only if it meets all restrictions associated with at least one of the super types. More generally, an instance of a type that has a union type in its type hierarchy is valid if and only if it is a valid instance of at least one of the super types obtained by expanding all unions in that type hierarchy. Such an instance is deserialized by performing this expansion and then matching the instance against all the super types, starting from the left-most and proceeding to the right; the first successfully-matching base type is used to deserialize the instance.

The following example defines two types and a third type which is a union of those two types.

```
types:
  CatOrDog:
    type: Cat | Dog # elements: Cat or Dog
  Cat:
    type: object
    properties:
      name: string
      color: string
  Dog:
    type: object
    properties:
      name: string
      fangs: string
```

The following example of an instance of type `CatOrDog` is valid:

```
CatOrDog: # follows restrictions applied to the type 'Cat'
  name: Musia,
  color: brown
```

Imagine a more complex example of a union type used in a multiple inheritance type expression:

```
types:
   HasHome:
     type: object
     properties:
       homeAddress: string
```

```
  Cat:
    type: object
    properties:
      name: string
      color: string
  Dog:
    type: object
    properties:
      name: string
      fangs: string
  HomeAnimal: [ HasHome ,  Dog | Cat ]
```

In this case, type `HomeAnimal` has two super types, `HasHome` and an anonymous union type, defined by the following type expression: `Dog | Cat`.

Validating the `HomeAnimal` type involves validating the types derived from each of the super types and the types of each element in the union type. In this particular case, the processor MUST test that types `[HasHome, Dog]` and `[HasHome, Cat]`are valid types.

If you are extending from two union types a processor MUST perform validations for every possible combination. For example, to validate the `HomeAnimal` type shown below, the processor MUST test the six possible combinations: `[HasHome, Dog ]`, `[HasHome, Cat ]`, `[HasHome, Parrot]`, `[IsOnFarm, Dog ]`, `[IsOnFarm, Cat ]`, and `[IsOnFarm, Parrot]`.

```
types:
  HomeAnimal: [ HasHome | IsOnFarm ,  Dog | Cat | Parrot ]
```

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[72]Using XML and JSON Schema

RAML allows the use of XML and JSON schemas to describe the body of an API request or response by integrating the schemas into its data type system.

The following examples show how to include an external JSON schema into a root-level type definition and a body declaration.

```
types:
  Person: !include person.json
```

```
/person:
  get:
    responses:
      200:
        body:
          application/json:
            type: !include person.json
```

A RAML processor MUST NOT allow types that define an XML or JSON schema to participate in type inheritance or specialization, or effectively in any type expression[73]. Therefore, you cannot define sub-types of these types to declare new properties, add restrictions, set facets, or declare facets. You can, however,

[72] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#using-xml-and-json-schema
[73] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#type-expressions

create simple type wrappers that add annotations, examples, display name, or a description.

The following example shows a valid declaration.

```
types:
  Person:
    type: !include person.json
    description: this is a schema describing person
```

The following example shows an invalid declaration of a type that inherits the characteristics of a JSON schema and adds additional properties.

```
types:
  Person:
    type: !include person.json
    properties: # invalid
      single: boolean
```

Another invalid case is shown in the following example of the type `Person` being used as a property type.

```
types:
  Person:
    type: !include person.json
    description: this is a schema describing person
  Board:
    properties:
      members: Person[] # invalid use of type expression '[]' and as a
 property type
```

A RAML processor MUST be able to interpret and apply JSON Schema and XML Schema.

An XML schema, or JSON schema, MUST NOT be used where the media type does not allow XML-formatted data, or JSON-formatted data, respectively. XML and JSON schemas are also forbidden in any declaration of query parameters, query string, URI parameters, and headers.

The nodes "schemas" and "types", as well as "schema" and "type", are mutually exclusive and synonymous for compatibility with RAML 0.8. API definitions should use "types" and "type", as "schemas" and "schema" are deprecated and might be removed in a future RAML version.

<svg aria-hidden="true" class="octicon octicon-link" height="16" version="1.1" viewBox="0 0 16 16" width="16"><path fill-rule="evenodd" d="M4 9h1v1H4c-1.5 0-3-1.69-3-3.5S2.55 3 4 3h4c1.45 0 3 1.69 3 3.5 0 1.41-.91 2.72-2 3.25V8.59c.58-.45 1-1.27 1-2.09C10 5.22 8.98 4 8 4H4c-.98 0-2 1.22-2 2.5S3 9 4 9zm9-3h-1v1h1c1 0 2 1.22 2 2.5S13.98 12 13 12H9c-.98 0-2-1.22-2-2.5 0-.83.42-1.64 1-2.09V6.25c-1.09.53-2 1.84-2 3.25C6 11.31 7.55 13 9 13h4c1.45 0 3-1.69 3-3.5S14.5 6 13 6z"></path></svg>[74]References to Inner Elements

Sometimes it is necessary to refer to an element defined in a schema. RAML supports that by using URL fragments as shown in the example below.

```
type: !include elements.xsd#Foo
```

When referencing an inner element of a schema, a RAML processor MUST validate an instance against that particular element. The RAML specification supports referencing any inner elements in JSON schemas that are valid schemas, any globally defined elements, and complex types in XML schemas. There are only a few restrictions:

- Validation of any XML or JSON instance against inner elements follows the same restrictions as the validation against a regular XML or JSON schema.
- Referencing complex types inside an XSD is valid to determine the structure of an XML instance, but since complex types do not define a name for the top-level XML element, these types cannot be used for serializing an XML instance.

[74] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/#references-to-inner-elements