# Swagger RESTful API Documentation Specification

### Version 2.0

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

## Introduction

Swagger™ is a project used to describe and document RESTful APIs.

The Swagger specification defines a set of files required to describe such an API. These files can then be used by the Swagger-UI project to display the API and Swagger-Codegen to generate clients in various languages. Additional utilities can also take advantage of the resulting files, such as testing tools.

## Revision History

| Version | Date | Notes |
|---|---|---|
| 2.0 | 2014-09-08 | Release of Swagger 2.0 |
| 1.2 | 2014-03-14 | Initial release of the formal document. |
| 1.1 | 2012-08-22 | Release of Swagger 1.1 |
| 1.0 | 2011-08-10 | First release of the Swagger Specification |

## Definitions

#### Path Templating

Path templating refers to the usage of curly braces ({}) to mark a section of a URL path as replaceable using path parameters.

#### Mime Types

Mime type definitions are spread across several resources. The mime type definitions should be in compliance with RFC 6838.

Some examples of possible mime type definitions:

```
    text/plain; charset=utf-8
    application/json
    application/vnd.github+json
    application/vnd.github.v3+json
    application/vnd.github.v3.raw+json
    application/vnd.github.v3.text+json
    application/vnd.github.v3.html+json
    application/vnd.github.v3.full+json
    application/vnd.github.v3.diff
    application/vnd.github.v3.patch
```

#### HTTP Status Codes

The HTTP Status Codes are used to indicate the status of the executed operation. The available status codes are described by RFC 7231 and in the IANA Status Code Registry.

## Specification

### Format

The files describing the RESTful API in accordance with the Swagger specification are represented as JSON objects and conform to the JSON standards. YAML, being a superset of JSON, can be used as well to represent a Swagger specification file.

For example, if a field is said to have an array value, the JSON array representation will be used:

```
{
   "field" : [...]
}
```

While the API is described using JSON it does not impose a JSON input/output to the API itself.

All field names in the specification are **case sensitive**.

The schema exposes two types of fields. Fixed fields, which have a declared name, and Patterned fields, which declare a regex pattern for the field name. Patterned fields can have multiple occurrences as long as each has a unique name.

## File Structure

The Swagger representation of the API is made of a single file. However, parts of the definitions can be split into separate files, at the discretion of the user. This is applicable for `$ref` fields in the specification as follows from the JSON Schema definitions.

By convention, the Swagger specification file is named `swagger.json` .

## Data Types

Primitive data types in the Swagger Specification are based on the types supported by the JSON-Schema Draft 4. Models are described using the Schema Object which is a subset of JSON Schema Draft 4.

An additional primitive data type `"file"` is used by the Parameter Object and the Response Object to set the parameter type or the response as being a file.

Primitives have an optional modifier property `format` . Swagger uses several known formats to more finely define the data type being used. However, the `format` property is an open `string` -valued property, and can have any value to support documentation needs. Formats such as `"email"` , `"uuid"` , etc., can be used even though they are not defined by this specification. Types that are not accompanied by a `format` property follow their definition from the JSON Schema (except for `file` type which is defined above). The formats defined by the Swagger Specification are:

| Common Name | `type` | `format` | Comments |
|---|---|---|---|
| integer | `integer` | `int32` | signed 32 bits |
| long | `integer` | `int64` | signed 64 bits |
| float | `number` | `float` | |
| double | `number` | `double` | |
| string | `string` | | |
| byte | `string` | `byte` | |
| boolean | `boolean` | | |
| date | `string` | `date` | As defined by `full-date` - RFC3339 |
| dateTime | `string` | `date-time` | As defined by `date-time` - RFC3339 |
| password | `string` | `password` | Used to hint UIs the input needs to be obscured. |

## Schema

### Swagger Object

This is the root document object for the API specification. It combines what previously was the Resource Listing and API Declaration (version 1.2 and earlier) together into one document.

#### Fixed Fields

| Field Name | Type | Description |
|---|---|---|
| swagger | `string` | **Required.** Specifies the Swagger Specification version being used. It can be used by the Swagger UI and other clients to interpret the API listing. The value MUST be `"2.0"` . |
| info | Info Object | **Required.** Provides metadata about the API. The metadata can be used by the clients if needed. |
| host | `string` | The host (name or ip) serving the API. This MUST be the host only and does not include the scheme nor sub-paths. It MAY include a port. If the `host` is not included, the host serving the documentation is to be used (including the port). The `host` does not support path templating. |
| basePath | `string` | The base path on which the API is served, which is relative to the `host` . If it is not included, the API is served directly under the `host` . The value MUST start with a leading slash ( `/` ). The `basePath` does not support path templating. |
| schemes | [ `string` ] | The transfer protocol of the API. Values MUST be from the list: `"http"` , `"https"` , `"ws"` , `"wss"` . If the `schemes` is not included, the default scheme to be used is the one used to access the specification. |
| consumes | [ `string` ] | A list of MIME types the APIs can consume. This is global to all APIs but can be overridden on specific API calls. Value MUST be as described under Mime Types. |
| produces | [ `string` ] | A list of MIME types the APIs can produce. This is global to all APIs but can be overridden on specific API calls. Value MUST be as described under Mime Types. |

| | | |
|---|---|---|
| paths | Paths Object | **Required.** The available paths and operations for the API. |
| definitions | Definitions Object | An object to hold data types produced and consumed by operations. |
| parameters | Parameters Definitions Object | An object to hold parameters that can be used across operations. This property *does not* define global parameters for all operations. |
| responses | Responses Definitions Object | An object to hold responses that can be used across operations. This property *does not* define global responses for all operations. |
| securityDefinitions | Security Definitions Object | Security scheme definitions that can be used across the specification. |
| security | [Security Requirement Object] | A declaration of which security schemes are applied for the API as a whole. The list of values describes alternative security schemes that can be used (that is, there is a logical OR between the security requirements). Individual operations can override this definition. |
| tags | [Tag Object] | A list of tags used by the specification with additional metadata. The order of the tags can be used to reflect on their order by the parsing tools. Not all tags that are used by the Operation Object must be declared. The tags that are not declared may be organized randomly or based on the tools' logic. Each tag name in the list MUST be unique. |
| externalDocs | External Documentation | Additional external documentation. |

| | | |
|---|---|---|
| | Object | |

## Info Object

The object provides metadata about the API. The metadata can be used by the clients if needed, and can be presented in the Swagger-UI for convenience.

### Fixed Fields

| Field Name | Type | Description |
|---|---|---|
| title | `string` | **Required.** The title of the application. |
| description | `string` | A short description of the application. GFM syntax can be used for rich text representation. |
| termsOfService | `string` | The Terms of Service for the API. |
| contact | Contact Object | The contact information for the exposed API. |
| license | License Object | The license information for the exposed API. |
| version | `string` | **Required** Provides the version of the application API (not to be confused with the specification version). |

### Patterned Objects

| Field Pattern | Type | Description |
|---|---|---|

| | | |
|---|---|---|
| ^x- | Any | Allows extensions to the Swagger Schema. The field name MUST begin with `x-` , for example, `x-internal-id` . The value can be `null` , a primitive, an array or an object. See Vendor Extensions for further details. |

**Info Object Example:**

```
{
  "title": "Swagger Sample App",
  "description": "This is a sample server Petstore server.",
  "termsOfService": "http://swagger.io/terms/",
  "contact": {
    "name": "API Support",
    "url": "http://www.swagger.io/support",
    "email": "support@swagger.io"
  },
  "license": {
    "name": "Apache 2.0",
    "url": "http://www.apache.org/licenses/LICENSE-2.0.html"
  },
  "version": "1.0.1"
}
```

```
title: Swagger Sample App
description: This is a sample server Petstore server.
termsOfService: http://swagger.io/terms/
contact:
  name: API Support
  url: http://www.swagger.io/support
  email: support@swagger.io
license:
  name: Apache 2.0
```

```
    url: http://www.apache.org/licenses/LICENSE-2.0.html
  version: 1.0.1
```

## Contact Object

Contact information for the exposed API.

#### Fixed Fields

| Field Name | Type | Description |
|---|---|---|
| name | `string` | The identifying name of the contact person/organization. |
| url | `string` | The URL pointing to the contact information. MUST be in the format of a URL. |
| email | `string` | The email address of the contact person/organization. MUST be in the format of an email address. |

**Contact Object Example:**

```
{
  "name": "API Support",
  "url": "http://www.swagger.io/support",
  "email": "support@swagger.io"
}
```

```
name: API Support
url: http://www.swagger.io/support
email: support@swagger.io
```

## License Object

License information for the exposed API.

#### Fixed Fields

| Field Name | Type | Description |
|---|---|---|
| name | `string` | **Required.** The license name used for the API. |
| url | `string` | A URL to the license used for the API. MUST be in the format of a URL. |

#### License Object Example:

```
{
  "name": "Apache 2.0",
  "url": "http://www.apache.org/licenses/LICENSE-2.0.html"
}
```

```
name: Apache 2.0
url: http://www.apache.org/licenses/LICENSE-2.0.html
```

## Paths Object

Holds the relative paths to the individual endpoints. The path is appended to the `basePath` in order to construct the full URL. The Paths may be empty, due to ACL constraints.

#### Patterned Fields

| Field Pattern | Type | Description |
|---|---|---|
| /{path} | Path Item Object | A relative path to an individual endpoint. The field name MUST begin with a slash. The path is appended to the `basePath` in order to construct the full URL. Path templating is allowed. |
| ^x- | Any | Allows extensions to the Swagger Schema. The field name MUST begin with `x-`, for example, `x-internal-id`. The value can be `null`, a primitive, an array or an object. See Vendor Extensions for further details. |

**Paths Object Example**

```
{
  "/pets": {
    "get": {
      "description": "Returns all pets from the system that the user has access to",
      "produces": [
        "application/json"
      ],
      "responses": {
        "200": {
          "description": "A list of pets.",
          "schema": {
            "type": "array",
            "items": {
              "$ref": "#/definitions/pet"
            }
          }
        }
      }
      }
    }
  }
```

```
/pets:
  get:
    description: Returns all pets from the system that the user has access to
    produces:
    - application/json
    responses:
      200:
        description: A list of pets.
        schema:
          type: array
          items:
            $ref: '#/definitions/pet'
```

### Path Item Object

Describes the operations available on a single path. A Path Item may be empty, due to ACL constraints. The path itself is still exposed to the documentation viewer but they will not know which operations and parameters are available.

#### Fixed Fields

| Field Name | Type | Description |
|---|---|---|
| $ref | `string` | Allows for an external definition of this path item. The referenced structure MUST be in the format of a Path Item Object. If there are conflicts between the referenced definition and this Path Item's definition, the behavior is *undefined*. |

| | | |
|---|---|---|
| get | Operation Object | A definition of a GET operation on this path. |
| put | Operation Object | A definition of a PUT operation on this path. |
| post | Operation Object | A definition of a POST operation on this path. |
| delete | Operation Object | A definition of a DELETE operation on this path. |
| options | Operation Object | A definition of a OPTIONS operation on this path. |
| head | Operation Object | A definition of a HEAD operation on this path. |
| patch | Operation Object | A definition of a PATCH operation on this path. |
| parameters | [Parameter Object \| Reference Object] | A list of parameters that are applicable for all the operations described under this path. These parameters can be overridden at the operation level, but cannot be removed there. The list MUST NOT include duplicated parameters. A unique parameter is defined by a combination of a name and location. The list can use the Reference Object to link to parameters that are defined at the Swagger Object's parameters. There can be one "body" parameter at most. |

**Patterned Fields**

| Field Pattern | Type | Description |
|---|---|---|
| ^x- | Any | Allows extensions to the Swagger Schema. The field name MUST begin with `x-`, for example, `x-internal-id`. The value can be `null`, a primitive, an array or an object. See Vendor Extensions for further details. |

#### Path Item Object Example

```
{
  "get": {
    "description": "Returns pets based on ID",
    "summary": "Find pets by ID",
    "operationId": "getPetsById",
    "produces": [
      "application/json",
      "text/html"
    ],
    "responses": {
      "200": {
        "description": "pet response",
        "schema": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/Pet"
          }
        }
      },
      "default": {
        "description": "error payload",
        "schema": {
          "$ref": "#/definitions/ErrorModel"
          }
        }
      }
    },
    "parameters": [
      {
        "name": "id",
        "in": "path",
        "description": "ID of pet to use",
        "required": true,
        "type": "array",
        "items": {
          "type": "string"
        },
        "collectionFormat": "csv"
      }
    ]
  }
```

```
get:
  description: Returns pets based on ID
  summary: Find pets by ID
  operationId: getPetsById
  produces:
  - application/json
  - text/html
  responses:
    200:
      description: pet response
      schema:
        type: array
        items:
          $ref: '#/definitions/Pet'
    default:
      description: error payload
      schema:
        $ref: '#/definitions/ErrorModel'
parameters:
- name: id
  in: path
  description: ID of pet to use
  required: true
  type: array
  items:
    type: string
  collectionFormat: csv
```

### Operation Object

Describes a single API operation on a path.

#### Fixed Fields

| Field Name | Type | Description |
|---|---|---|
| tags | [ `string` ] | A list of tags for API documentation control. Tags can be used for logical grouping of operations by resources or any other qualifier. |
| summary | `string` | A short summary of what the operation does. For maximum readability in the swagger-ui, this field SHOULD be less than 120 characters. |
| description | `string` | A verbose explanation of the operation behavior. GFM syntax can be used for rich text representation. |

| | | |
|---|---|---|
| externalDocs | External Documentation Object | Additional external documentation for this operation. |
| operationId | `string` | A friendly name for the operation. The id MUST be unique among all operations described in the API. Tools and libraries MAY use the operation id to uniquely identify an operation. |
| consumes | [ `string` ] | A list of MIME types the operation can consume. This overrides the `consumes` definition at the Swagger Object. An empty value MAY be used to clear the global definition. Value MUST be as described under Mime Types. |
| produces | [ `string` ] | A list of MIME types the operation can produce. This overrides the `produces` definition at the Swagger Object. An empty value MAY be used to clear the global definition. Value MUST be as described under Mime Types. |
| parameters | [Parameter Object \| Reference Object] | A list of parameters that are applicable for this operation. If a parameter is already defined at the Path Item, the new definition will override it, but can never remove it. The list MUST NOT include duplicated parameters. A unique parameter is defined by a combination of a name and location. The list can use the Reference Object to link to parameters that are defined at the Swagger Object's parameters. There can be one "body" parameter at most. |
| responses | Responses Object | **Required.** The list of possible responses as they are returned from executing this operation. |
| schemes | [ `string` ] | The transfer protocol for the operation. Values MUST be from the list: `"http"` , `"https"` , `"ws"` , `"wss"` . The value overrides the Swagger Object `schemes` definition. |

|  |  |  |
|---|---|---|
| deprecated | `boolean` | Declares this operation to be deprecated. Usage of the declared operation should be refrained. Default value is `false` . |
| security | [Security Requirement Object] | A declaration of which security schemes are applied for this operation. The list of values describes alternative security schemes that can be used (that is, there is a logical OR between the security requirements). This definition overrides any declared top-level `security` . To remove a top-level security declaration, an empty array can be used. |

#### Patterned Objects

| Field Pattern | Type | Description |
|---|---|---|
| ^x- | Any | Allows extensions to the Swagger Schema. The field name MUST begin with `x-` , for example, `x-internal-id` . The value can be `null` , a primitive, an array or an object. See Vendor Extensions for further details. |

#### Operation Object Example

```
{
  "tags": [
    "pet"
  ],
  "summary": "Updates a pet in the store with form data",
  "description": "",
  "operationId": "updatePetWithForm",
  "consumes": [
    "application/x-www-form-urlencoded"
  ],
        "produces": [
          "application/json",
          "application/xml"
        ],
        "parameters": [
          {
            "name": "petId",
            "in": "path",
            "description": "ID of pet that needs to be updated",
            "required": true,
            "type": "string"
          },
          {
            "name": "name",
            "in": "formData",
            "description": "Updated name of the pet",
            "required": false,
            "type": "string"
          },
          {
            "name": "status",
            "in": "formData",
            "description": "Updated status of the pet",
            "required": false,
            "type": "string"
          }
        ],
        "responses": {
          "200": {
            "description": "Pet updated."
          },
          "405": {
            "description": "Invalid input"
          }
```

```
    },
    "security": [
      {
        "petstore_auth": [
          "write:pets",
          "read:pets"
        ]
      }
    ]
  }
```

```
tags:
- pet
summary: Updates a pet in the store with form data
description: ""
operationId: updatePetWithForm
consumes:
- application/x-www-form-urlencoded
produces:
- application/json
- application/xml
parameters:
- name: petId
  in: path
  description: ID of pet that needs to be updated
  required: true
  type: string
- name: name
  in: formData
  description: Updated name of the pet
  required: false
  type: string
  - name: status
    in: formData
    description: Updated status of the pet
    required: false
    type: string
  responses:
    200:
      description: Pet updated.
    405:
      description: Invalid input
  security:
  - petstore_auth:
    - write:pets
    - read:pets
```

## External Documentation Object

Allows referencing an external resource for extended documentation.

### Fixed Fields

| Field Name | Type | Description |
|---|---|---|
| description | `string` | A short description of the target documentation. GFM syntax can be used for rich text representation. |
| url | `string` | **Required.** The URL for the target documentation. Value MUST be in the format of a URL. |

### External Documentation Object Example

```json
{
  "description": "Find more info here",
  "url": "https://swagger.io"
}
```

```yaml
description: Find more info here
url: https://swagger.io
```

## Parameter Object

Describes a single operation parameter.

A unique parameter is defined by a combination of a name and location.

There are five possible parameter types.

- Path - Used together with Path Templating, where the parameter value is actually part of the operation's URL. This does not include the host or base path of the API. For example, in `/items/{itemId}` , the path parameter is `itemId` .
- Query - Parameters that are appended to the URL. For example, in `/items?id=###` , the query parameter is `id` .
- Header - Custom headers that are expected as part of the request.
- Body - The payload that's appended to the HTTP request. Since there can only be one payload, there can only be *one* body parameter. The name of the body parameter has no effect on the parameter itself and is used for documentation purposes only. Since Form parameters are also in the payload, body and form parameters cannot exist together for the same operation.
- Form - Used to describe the payload of an HTTP request when either `application/x-www-form-urlencoded` or `multipart/form-data` are used as the content type of the request (in Swagger's definition, the `consumes` property of an operation). This is the only parameter type that can be used to send files, thus supporting the `file` type. Since form

parameters are sent in the payload, they cannot be declared together with a body parameter for the same operation. Form parameters have a different format based on the content-type used (for further details, consult http://www.w3.org/TR/html401/interact/forms.html#h-17.13.4):

- `application/x-www-form-urlencoded` - Similar to the format of Query parameters but as a payload. For example, `foo=1&bar=swagger` - both `foo` and `bar` are form parameters. This is normally used for simple parameters that are being transferred.
- `multipart/form-data` - each parameter takes a section in the payload with an internal header. For example, for the header `Content-Disposition: form-data; name="submit-name"` the name of the parameter is `submit-name`. This type of form parameters is more commonly used for file transfers.

#### Fixed Fields

| Field Name | Type | Description |
|---|---|---|
| name | `string` | **Required.** The name of the parameter. Parameter names are *case sensitive*.<br>• If `in` is `"path"`, the `name` field MUST correspond to the associated path segment from the path field in the Paths Object. See Path Templating for further information.<br>• For all other cases, the `name` corresponds to the parameter name used based on the `in` property. |
| in | `string` | **Required.** The location of the parameter. Possible values are "query", "header", "path", "formData" or "body". |
| description | `string` | A brief description of the parameter. This could contain examples of use. GFM syntax can be used for rich text representation. |
| | | Determines whether this parameter is mandatory. If the parameter is `in` "path", this |

| | | |
|---|---|---|
| required | `boolean` | property is **required** and its value MUST be `true` . Otherwise, the property MAY be included and its default value is `false` . |

If `in` is `"body"` :

| Field Name | Type | Description |
|---|---|---|
| schema | Schema Object | **Required.** The schema defining the type used for the body parameter. |

If `in` is any value other than `"body"` :

| Field Name | Type | Description |
|---|---|---|
| type | `string` | **Required.** The type of the parameter. Since the parameter is not located at the request body, it is limited to simple types (that is, not an object). The value MUST be one of `"string"` , `"number"` , `"integer"` , `"boolean"` , `"array"` or `"file"` . If `type` is `"file"` , the `consumes` MUST be either `"multipart/form-data"` or `"application/x-www-form-urlencoded"` and the parameter MUST be `in` `"formData"` . |
| format | `string` | The extending format for the previously mentioned `type` . See Data Type Formats for further details. |
| allowEmptyValue | `boolean` | Sets the ability to pass empty-valued parameters. This is valid only for either `query` or `formData` parameters and allows you to send a parameter with a name only or an empty value. Default value is `false` . |
| items | Items Object | **Required if `type` is "array".** Describes the type of items in the array. |

| | | |
|---|---|---|
| collectionFormat | `string` | Determines the format of the array if type array is used. Possible values are:<br>• `csv` - comma separated values `foo,bar`.<br>• `ssv` - space separated values `foo bar`.<br>• `tsv` - tab separated values `foo\tbar`.<br>• `pipes` - pipe separated values `foo\|bar`.<br>• `multi` - corresponds to multiple parameter instances instead of multiple values for a single instance `foo=bar&foo=baz`. This is valid only for parameters `in` "query" or "formData".<br><br>Default value is `csv`. |
| default | * | Sets a default value to the parameter. See http://json-schema.org/latest/json-schema-validation.html#anchor101. Unlike JSON Schema this value MUST conform to the defined `type` for this parameter. |
| maximum | `number` | See http://json-schema.org/latest/json-schema-validation.html#anchor17. |
| exclusiveMaximum | `boolean` | See http://json-schema.org/latest/json-schema-validation.html#anchor17. |
| minimum | `number` | See http://json-schema.org/latest/json-schema-validation.html#anchor21. |
| exclusiveMinimum | `boolean` | See http://json-schema.org/latest/json-schema-validation.html#anchor21. |
| maxLength | `integer` | See http://json-schema.org/latest/json-schema-validation.html#anchor26. |
| minLength | `integer` | See http://json-schema.org/latest/json-schema-validation.html#anchor29. |
| pattern | `string` | See http://json-schema.org/latest/json-schema-validation.html#anchor33. |
| maxItems | `integer` | See http://json-schema.org/latest/json-schema-validation.html#anchor42. |
| minItems | `integer` | See http://json-schema.org/latest/json-schema-validation.html#anchor45. |

| | | |
|---|---|---|
| uniqueItems | `boolean` | See http://json-schema.org/latest/json-schema-validation.html#anchor49. |
| enum | [*] | See http://json-schema.org/latest/json-schema-validation.html#anchor76. |
| multipleOf | `number` | See http://json-schema.org/latest/json-schema-validation.html#anchor14. |

#### Patterned Fields

| Field Pattern | Type | Description |
|---|---|---|
| ^x- | Any | Allows extensions to the Swagger Schema. The field name MUST begin with `x-`, for example, `x-internal-id`. The value can be `null`, a primitive, an array or an object. See Vendor Extensions for further details. |

#### Parameter Object Examples

##### Body Parameters

A body parameter with a referenced schema definition (normally for a model definition):

```
{
  "name": "user",
  "in": "body",
  "description": "user to add to the system",
  "required": true,
  "schema": {
    "$ref": "#/definitions/User"
  }
}
```

```yaml
name: user
in: body
description: user to add to the system
required: true
schema:
  $ref: '#/definitions/User'
```

A body parameter that is an array of string values:

```json
{
  "name": "user",
  "in": "body",
  "description": "user to add to the system",
  "required": true,
  "schema": {
    "type": "array",
    "items": {
      "type": "string"
    }
  }
}
```

```yaml
name: user
in: body
description: user to add to the system
required: true
schema:
  type: array
  items:
    type: string
```

### Other Parameters

A header parameter with an array of 64 bit integer numbers:

```
{
  "name": "token",
  "in": "header",
  "description": "token to be passed as a header",
  "required": true,
  "type": "array",
  "items": {
    "type": "integer",
    "format": "int64"
  },
  "collectionFormat": "csv"
}
```

```
name: token
in: header
description: token to be passed as a header
required: true
type: array
items:
  type: integer
  format: int64
collectionFormat: csv
```

A path parameter of a string value:

```
{
    "name": "username",
    "in": "path",
    "description": "username to fetch",
    "required": true,
    "type": "string"
  }
```

```
name: username
in: path
description: username to fetch
required: true
type: string
```

An optional query parameter of a string value, allowing multiple values by repeating the query parameter:

```
{
  "name": "id",
  "in": "query",
  "description": "ID of the object to fetch",
  "required": false,
  "type": "array",
  "items": {
    "type": "string"
  },
  "collectionFormat": "multi"
}
```

```
name: id
in: query
description: ID of the object to fetch
```

```yaml
required: false
type: array
items:
  type: string
collectionFormat: multi
```

A form data with file type for a file upload:

```json
{
  "name": "avatar",
  "in": "formData",
  "description": "The avatar of the user",
  "required": true,
  "type": "file"
}
```

```yaml
name: avatar
in: formData
description: The avatar of the user
required: true
type: file
```

## Items Object

An limited subset of JSON-Schema's items object. It is used by parameter definitions that are not located `in` `"body"`.

### Fixed Fields

| Field Name | Type | Description |
|---|---|---|

| | | |
|---|---|---|
| type | `string` | **Required.** The internal type of the array. The value MUST be one of `"string"`, `"number"`, `"integer"`, `"boolean"`, or `"array"`. Files and models are not allowed. |
| format | `string` | The extending format for the previously mentioned `type`. See Data Type Formats for further details. |
| items | Items Object | **Required if `type` is "array".** Describes the type of items in the array. |
| collectionFormat | `string` | Determines the format of the array if type array is used. Possible values are:<br>• `csv` - comma separated values `foo,bar`.<br>• `ssv` - space separated values `foo bar`.<br>• `tsv` - tab separated values `foo\tbar`.<br>• `pipes` - pipe separated values `foo\|bar`.<br><br>Default value is `csv`. |
| default | * | Sets a default value to the data type. See http://json-schema.org/latest/json-schema-validation.html#anchor101. Unlike JSON Schema this value MUST conform to the defined `type` for the data type. |
| maximum | `number` | See http://json-schema.org/latest/json-schema-validation.html#anchor17. |
| exclusiveMaximum | `boolean` | See http://json-schema.org/latest/json-schema-validation.html#anchor17. |
| minimum | `number` | See http://json-schema.org/latest/json-schema-validation.html#anchor21. |
| exclusiveMinimum | `boolean` | See http://json-schema.org/latest/json-schema-validation.html#anchor21. |
| maxLength | `integer` | See http://json-schema.org/latest/json-schema-validation.html#anchor26. |

|  |  |  |
|---|---|---|
| minLength | `integer` | See http://json-schema.org/latest/json-schema-validation.html#anchor29. |
| pattern | `string` | See http://json-schema.org/latest/json-schema-validation.html#anchor33. |
| maxItems | `integer` | See http://json-schema.org/latest/json-schema-validation.html#anchor42. |
| minItems | `integer` | See http://json-schema.org/latest/json-schema-validation.html#anchor45. |
| uniqueItems | `boolean` | See http://json-schema.org/latest/json-schema-validation.html#anchor49. |
| enum | [*] | See http://json-schema.org/latest/json-schema-validation.html#anchor76. |
| multipleOf | `number` | See http://json-schema.org/latest/json-schema-validation.html#anchor14. |

#### Items Object Examples

Items must be of type string and have the minimum length of 2 characters:

```
{
    "type": "string",
    "minLength": 2
}
```

```
type: string
minLength: 2
```

An array of arrays, the internal array being of type integer, numbers must be between 0 and 63 (inclusive):

```
{
    "type": "array",
        "items": {
            "type": "integer",
            "minimum": 0,
            "maximum": 63
        }
    }
```

```
type: array
items:
  type: integer
  minimum: 0
  maximum: 63
```

### Responses Object

A container for the expected responses of an operation. The container maps a HTTP response code to the expected response. It is not expected from the documentation to necessarily cover all possible HTTP response codes, since they may not be known in advance. However, it is expected from the documentation to cover a successful operation response and any known errors.

The `default` can be used a default response object for all HTTP codes that are not covered individually by the specification.

The `Responses Object` MUST contain at least one response code, and it SHOULD be the response for a successful operation call.

#### Fixed Fields

| Field Name | Type | Description |
|---|---|---|

| | | |
|---|---|---|
| default | Response Object \| Reference Object | The documentation of responses other than the ones declared for specific HTTP response codes. It can be used to cover undeclared responses. Reference Object can be used to link to a response that is defined at the Swagger Object's responses section. |

#### Patterned Fields

| Field Pattern | Type | Description |
|---|---|---|
| {HTTP Status Code} | Response Object \| Reference Object | Any HTTP status code can be used as the property name (one property per HTTP status code). Describes the expected response for that HTTP status code. Reference Object can be used to link to a response that is defined at the Swagger Object's responses section. |
| ^x- | Any | Allows extensions to the Swagger Schema. The field name MUST begin with `x-`, for example, `x-internal-id`. The value can be `null`, a primitive, an array or an object. See Vendor Extensions for further details. |

#### Responses Object Example

A 200 response for successful operation and a default response for others (implying an error):

```
{
  "200": {
    "description": "a pet to be returned",
    "schema": {
      "$ref": "#/definitions/Pet"
    }
  },
    "default": {
      "description": "Unexpected error",
      "schema": {
        "$ref": "#/definitions/ErrorModel"
      }
    }
  }
```

```
  200:
    description: a pet to be returned
    schema:
      $ref: '#/definitions/Pet'
  default:
    description: Unexpected error
    schema:
      $ref: '#/definitions/ErrorModel'
```

## Response Object

Describes a single response from an API Operation.

### Fixed Fields

| Field Name | Type | Description |
|---|---|---|
| description | `string` | **Required.** A short description of the response. GFM syntax can be used for rich text representation. |
| | | A definition of the response structure. It can be a primitive, an array or an object. If this field |

|  |  |  |
|---|---|---|
| schema | Schema Object | does not exist, it means no content is returned as part of the response. As an extension to the Schema Object, its root `type` value may also be `"file"`. This SHOULD be accompanied by a relevant `produces` mime-type. |
| headers | Headers Object | A list of headers that are sent with the response. |
| examples | Example Object | An example of the response message. |

#### Response Object Examples

Response of an array of a complex type:

```
{
  "description": "A complex object array response",
  "schema": {
    "type": "array",
    "items": {
      "$ref": "#/definitions/VeryComplexType"
    }
  }
}
```

```
description: A complex object array response
schema:
  type: array
  items:
    $ref: '#/definitions/VeryComplexType'
```

Response with a string type:

```json
{
  "description": "A simple string response",
  "schema": {
    "type": "string"
  }
}
```

```yaml
description: A simple string response
schema:
  type: string
```

Response with headers:

```json
{
  "description": "A simple string response",
  "schema": {
    "type": "string"
  },
  "headers": {
    "X-Rate-Limit-Limit": {
      "description": "The number of allowed requests in the current period",
      "type": "integer"
    },
    "X-Rate-Limit-Remaining": {
      "description": "The number of remaining requests in the current period",
      "type": "integer"
    },
    "X-Rate-Limit-Reset": {
        "description": "The number of seconds left in the current period",
        "type": "integer"
      }
    }
  }
```

```
  description: A simple string response
  schema:
    type: string
  headers:
    X-Rate-Limit-Limit:
      description: The number of allowed requests in the current period
      type: integer
    X-Rate-Limit-Remaining:
      description: The number of remaining requests in the current period
      type: integer
    X-Rate-Limit-Reset:
      description: The number of seconds left in the current period
      type: integer
```

Response with no return value:

```
  {
    "description": "object created"
  }
```

```
  description: object created
```

## Headers Object

Lists the headers that can be sent as part of a response.

### Patterned Fields

| Field Pattern | Type | Description |
|---|---|---|
| {name} | Header Object | The name of the property corresponds to the name of the header. The value describes the type of the header. |

### Headers Object Example

Rate-limit headers:

```
{
    "X-Rate-Limit-Limit": {
        "description": "The number of allowed requests in the current period",
        "type": "integer"
    },
    "X-Rate-Limit-Remaining": {
        "description": "The number of remaining requests in the current period",
        "type": "integer"
    },
    "X-Rate-Limit-Reset": {
        "description": "The number of seconds left in the current period",
        "type": "integer"
    }
}
```

```
X-Rate-Limit-Limit:
  description: The number of allowed requests in the current period
  type: integer
X-Rate-Limit-Remaining:
  description: The number of remaining requests in the current period
  type: integer
X-Rate-Limit-Reset:
  description: The number of seconds left in the current period
  type: integer
```

## Example Object

Allows sharing examples for operation responses.

#### Patterned Fields

| Field Pattern | Type | Description |
|---|---|---|
| {mime type} | Any | The name of the property MUST be one of the Operation `produces` values (either implicit or inherited). The value SHOULD be an example of what such a response would look like. |

#### Example Object Example

Example response for application/json mimetype of a Pet data type:

```
{
  "application/json": {
    "name": "Puma",
    "type": "Dog",
    "color": "Black",
      "gender": "Female",
      "breed": "Mixed"
    }
  }
```

```
application/json:
  name: Puma
  type: Dog
  color: Black
  gender: Female
  breed: Mixed
```

## Header Object

| Field Name | Type | Description |
|---|---|---|
| description | `string` | A short description of the header. |
| type | `string` | **Required.** The type of the object. The value MUST be one of `"string"` , `"number"` , `"integer"` , `"boolean"` , or `"array"` . |
| format | `string` | The extending format for the previously mentioned `type` . See Data Type Formats for further details. |
| items | Items Object | **Required if `type` is "array".** Describes the type of items in the array. |
| | | Determines the format of the array if type array is used. Possible values are: <br>• `csv` - comma separated values `foo,bar` . <br>• `ssv` - space separated values `foo bar` . |

| | | |
|---|---|---|
| collectionFormat | `string` | <ul><li>`tsv` - tab separated values `foo\tbar`.</li><li>`pipes` - pipe separated values `foo\|bar`.</li></ul> Default value is `csv`. |
| default | * | Sets a default value for the header. See http://json-schema.org/latest/json-schema-validation.html#anchor101. Unlike JSON Schema this value MUST conform to the defined `type` for the header. |
| maximum | `number` | See http://json-schema.org/latest/json-schema-validation.html#anchor17. |
| exclusiveMaximum | `boolean` | See http://json-schema.org/latest/json-schema-validation.html#anchor17. |
| minimum | `number` | See http://json-schema.org/latest/json-schema-validation.html#anchor21. |
| exclusiveMinimum | `boolean` | See http://json-schema.org/latest/json-schema-validation.html#anchor21. |
| maxLength | `integer` | See http://json-schema.org/latest/json-schema-validation.html#anchor26. |
| minLength | `integer` | See http://json-schema.org/latest/json-schema-validation.html#anchor29. |
| pattern | `string` | See http://json-schema.org/latest/json-schema-validation.html#anchor33. |
| maxItems | `integer` | See http://json-schema.org/latest/json-schema-validation.html#anchor42. |
| minItems | `integer` | See http://json-schema.org/latest/json-schema-validation.html#anchor45. |
| uniqueItems | `boolean` | See http://json-schema.org/latest/json-schema-validation.html#anchor49. |
| enum | [*] | See http://json-schema.org/latest/json-schema-validation.html#anchor76. |
| multipleOf | `number` | See http://json-schema.org/latest/json-schema-validation.html#anchor14. |

#### Header Object Example

A simple header with of an integer type:

```json
{
  "description": "The number of allowed requests in the current period",
  "type": "integer"
}
```

```yaml
description: The number of allowed requests in the current period
type: integer
```

### Tag Object

Allows adding meta data to a single tag that is used by the Operation Object. It is not mandatory to have a Tag Object per tag used there.

#### Fixed Fields

| Field Name | Type | Description |
|---|---|---|
| name | `string` | **Required.** The name of the tag. |
| description | `string` | A short description for the tag. GFM syntax can be used for rich text representation. |
| externalDocs | External Documentation Object | Additional external documentation for this tag. |

#### Patterned Fields

| Field Pattern | Type | Description |
|---|---|---|
| ^x- | Any | Allows extensions to the Swagger Schema. The field name MUST begin with `x-`, for example, `x-internal-id`. The value can be `null`, a primitive, an array or an object. See Vendor Extensions for further details. |

#### Tag Object Example

```
{
    "name": "pet",
    "description": "Pets operations"
}
```

```
name: pet
description: Pets operations
```

### Reference Object

A simple object to allow referencing other definitions in the specification. It can be used to reference parameters and responses that are defined at the top level for reuse.

The Reference Object is a JSON Reference that uses a JSON Pointer as its value. For this specification, only canonical dereferencing is supported.

#### Fixed Fields

| Field Name | Type | Description |
|---|---|---|
| $ref | `string` | **Required.** The reference string. |

**Reference Object Example**

```
{
    "$ref": "#/definitions/Pet"
}
```

```
$ref: '#/definitions/Pet'
```

## Schema Object

The Schema Object allows the definition of input and output data types. These types can be objects, but also primitives and arrays. This object is based on the JSON Schema Specification Draft 4 and uses a predefined subset of it. On top of this subset, there are extensions provided by this specification to allow for more complete documentation.

Further information about the properties can be found in JSON Schema Core and JSON Schema Validation. Unless stated otherwise, the property definitions follow the JSON Schema specification as referenced here.

The following properties are taken directly from the JSON Schema definition and follow the same specifications:

- $ref - As a JSON Reference
- format (See Data Type Formats for further details)
- title
- description (GFM syntax can be used for rich text representation)
- default (Unlike JSON Schema, the value MUST conform to the defined type for the Schema Object)

- multipleOf
- maximum
- exclusiveMaximum
- minimum
- exclusiveMinimum
- maxLength
- minLength
- pattern
- maxItems
- minItems
- uniqueItems
- maxProperties
- minProperties
- required
- enum
- type

The following properties are taken from the JSON Schema definition but their definitions were adjusted to the Swagger Specification. Their definition is the same as the one from JSON Schema, only where the original definition references the JSON Schema definition, the Schema Object definition is used instead.

- items
- allOf
- properties
- additionalProperties

Other than the JSON Schema subset fields, the following fields may be used for further schema documentation.

**Fixed Fields**

| Field Name | Type | Description |
|---|---|---|
| discriminator | `string` | Adds support for polymorphism. The discriminator is the schema property name that is used to differentiate between other schema that inherit this schema. The property name used MUST be defined at this schema and it MUST be in the `required` property list. When used, the value MUST be the name of this schema or any schema that inherits it. |
| readOnly | `boolean` | Relevant only for Schema `"properties"` definitions. Declares the property as "read only". This means that it MAY be sent as part of a response but MUST NOT be sent as part of the request. Properties marked as `readOnly` being `true` SHOULD NOT be in the `required` list of the defined schema. Default value is `false` . |
| xml | XML Object | This MAY be used only on properties schemas. It has no effect on root schemas. Adds Additional metadata to describe the XML representation format of this property. |
| externalDocs | External Documentation Object | Additional external documentation for this schema. |
| example | Any | A free-form property to include a an example of an instance for this schema. |

#### Composition and Inheritance (Polymorphism)

Swagger allows combining and extending model definitions using the `allOf` property of JSON Schema, in effect offering model composition. `allOf` takes in an array of object definitions that are validated *independently* but together compose a single object.

While composition offers model extensibility, it does not imply a hierarchy between the models. To support polymorphism, Swagger adds the support of the `discriminator` field. When used, the `discriminator` will be the name of the property used to decide which schema definition is used to validate the structure of the model. As such, the `discriminator` field MUST be a required field. The value of the chosen property has to be the friendly name given to the model under the `definitions` property. As such, inline schema definitions, which do not have a given id, *cannot* be used in polymorphism.

#### XML Modeling

The xml property allows extra definitions when translating the JSON definition to XML. The XML Object contains additional information about the available options.

### Schema Object Examples

#### Primitive Sample

Unlike previous versions of Swagger, Schema definitions can be used to describe primitive and arrays as well.

```
{
    "type": "string",
    "format": "email"
}
```

```
type: string
format: email
```

#### Simple Model

```
{
  "type": "object",
    "required": [
      "name"
    ],
    "properties": {
      "name": {
        "type": "string"
      },
      "address": {
        "$ref": "#/definitions/Address"
      },
      "age": {
        "type": "integer",
        "format": "int32",
        "minimum": 0
      }
    }
  }
```

```
type: object
required:
- name
properties:
  name:
    type: string
  address:
    $ref: '#/definitions/Address'
  age:
    type: integer
    format: int32
    minimum: 0
```

## Model with Map/Dictionary Properties

For a simple string to string mapping:

```json
{
  "type": "object",
  "additionalProperties": {
    "type": "string"
  }
}
```

```yaml
type: object
additionalProperties:
  type: string
```

For a string to model mapping:

```json
{
  "type": "object",
  "additionalProperties": {
    "$ref": "#/definitions/ComplexModel"
  }
}
```

```yaml
type: object
additionalProperties:
  $ref: '#/definitions/ComplexModel'
```

#### Model with Example

```json
{
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64"
    },
    "name": {
      "type": "string"
    }
  },
  "required": [
    "name"
  ],
  "example": {
    "name": "Puma",
    "id": 1
  }
}
```

```yaml
properties:
  id:
    type: integer
    format: int64
  name:
    type: string
required:
- name
example:
  name: Puma
  id: 1
```

## Models with Composition

```
{
  "definitions": {
    "ErrorModel": {
      "type": "object",
      "required": [
        "message",
        "code"
      ],
      "properties": {
        "message": {
          "type": "string"
        },
        "code": {
          "type": "integer",
          "minimum": 100,
          "maximum": 600
        }
      }
    },
    "ExtendedErrorModel": {
      "allOf": [
        {
          "$ref": "#/definitions/ErrorModel"
        },
        {
          "type": "object",
          "required": [
            "rootCause"
          ],
          "properties": {
            "rootCause": {
              "type": "string"
            }
            }
          }
        ]
      }
    }
  }
```

```yaml
definitions:
  ErrorModel:
    type: object
    required:
    - message
    - code
    properties:
      message:
        type: string
      code:
        type: integer
        minimum: 100
        maximum: 600
  ExtendedErrorModel:
    allOf:
    - $ref: '#/definitions/ErrorModel'
    - type: object
      required:
      - rootCause
      properties:
        rootCause:
          type: string
```

## Models with Polymorphism Support

```
{
  "definitions": {
    "Pet": {
      "discriminator": "petType",
      "properties": {
        "name": {
          "type": "string"
        },
        "petType": {
          "type": "string"
        }
      },
      "required": [
        "name",
        "petType"
      ]
    }
  },
  "Cat": {
    "description": "A representation of a cat",
    "allOf": [
      {
        "$ref": "#/definitions/Pet"
      },
      {
        "properties": {
          "huntingSkill": {
            "type": "string",
            "description": "The measured skill for hunting",
            "default": "lazy",
            "enum": [
              "clueless",
              "lazy",
              "adventurous",
              "aggressive"
            ]
          }
        },
        "required": [
          "huntingSkill"
        ]
      }
    ]
  },
  "Dog": {
    "description": "A representation of a dog",
    "allOf": [
      {
        "$ref": "#/definitions/Pet"
      },
      {
        "properties": {
          "packSize": {
            "type": "integer",
            "format": "int32",
            "description": "the size of the pack the dog is from",
            "default": 0,
            "minimum": 0
          }
        },
        "required": [
          "packSize"
        ]
      }
    ]
  }
}
```

```yaml
definitions:
  Pet:
    discriminator: petType
    properties:
      name:
        type: string
      petType:
        type: string
    required:
    - name
    - petType
Cat:
  description: A representation of a cat
  allOf:
  - $ref: '#/definitions/Pet'
  - properties:
      huntingSkill:
        type: string
        description: The measured skill for hunting
        default: lazy
        enum:
        - clueless
        - lazy
        - adventurous
        - aggressive
    required:
    - huntingSkill
Dog:
  description: A representation of a dog
  allOf:
  - $ref: '#/definitions/Pet'
  - properties:
        packSize:
          type: integer
          format: int32
          description: the size of the pack the dog is from
          default: 0
          minimum: 0
      required:
      - packSize
```

### XML Object

A metadata object that allows for more fine-tuned XML model definitions.

When using arrays, XML element names are *not* inferred (for singular/plural forms) and the `name` property should be used to add that information. See examples for expected behavior.

#### Fixed Fields

| Field Name | Type | Description |
|---|---|---|
| name | `string` | Replaces the name of the element/attribute used for the described schema property. When defined within the Items Object ( `items` ), it will affect the name of the individual XML elements within the list. When defined alongside `type` being `array` (outside the `items` ), it will affect the wrapping element and only if `wrapped` is `true` . If `wrapped` is `false` , it will be ignored. |
| namespace | `string` | The URL of the namespace definition. Value SHOULD be in the form of a URL. |
| prefix | `string` | The prefix to be used for the name. |

| | | |
|---|---|---|
| attribute | `boolean` | Declares whether the property definition translates to an attribute instead of an element. Default value is `false` . |
| wrapped | `boolean` | MAY be used only for an array definition. Signifies whether the array is wrapped (for example, `<books><book/><book/></books>` ) or unwrapped ( `<book/><book/>` ). Default value is `false` . The definition takes effect only when defined alongside `type` being `array` (outside the `items` ). |

### XML Object Examples

The examples of the XML object definitions are included inside a property definition of a Schema Object with a sample of the XML representation of it.

#### No XML Element

Basic string property:

```
{
    "animals": {
        "type": "string"
    }
}
```

```
animals:
  type: string
```

```
<animals>...</animals>
```

Basic string array property ( `wrapped` is `false` by default):

```
{
    "animals": {
        "type": "array",
        "items": {
            "type": "string"
        }
    }
}
```

```
animals:
  type: array
  items:
    type: string
```

```
<animals>...</animals>
<animals>...</animals>
<animals>...</animals>
```

### XML Name Replacement

```
{
  "animals": {
    "type": "string",
    "xml": {
      "name": "animal"
    }
  }
}
```

```
animals:
  type: string
  xml:
    name: animal
```

```
<animal>...</animal>
```

### XML Attribute, Prefix and Namespace

In this example, a full model definition is shown.

```
{
  "Person": {
    "type": "object",
    "properties": {
      "id": {
        "type": "integer",
        "format": "int32",
        "xml": {
          "attribute": true
        },
        "name": {
          "type": "string",
          "xml": {
            "namespace": "http://swagger.io/schema/sample",
            "prefix": "sample"
          }
        }
        }
      }
    }
  }
```

```
  Person:
    type: object
    properties:
      id:
        type: integer
        format: int32
        xml:
          attribute: true
        name:
          type: string
          xml:
            namespace: http://swagger.io/schema/sample
            prefix: sample
```

```
  <Person id="123">
      <sample:name xlmns:sample="http://swagger.io/schema/sample">example</sample:name>
  </Person>
```

### XML Arrays

Changing the element names:

```
  {
    "animals": {
      "type": "array",
      "items": {
        "type": "string",
        "xml": {
          "name": "animal"
        }
      }
    }
  }
```

```
animals:
  type: array
  items:
    type: string
    xml:
      name: animal
```

```
<animal>value</animal>
<animal>value</animal>
```

The external `name` property has no effect on the XML:

```
{
  "animals": {
    "type": "array",
    "items": {
      "type": "string",
      "xml": {
        "name": "animal"
      }
    },
      "xml": {
        "name": "aliens"
      }
    }
  }
```

```
  animals:
    type: array
    items:
      type: string
      xml:
        name: animal
    xml:
      name: aliens
```

```
  <animal>value</animal>
  <animal>value</animal>
```

Even when the array is wrapped, if no name is explicitly defined, the same name will be used both internally and externally:

```
  {
    "animals": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "xml": {
        "wrapped": true
      }
    }
```

```
}
```

```
animals:
  type: array
  items:
    type: string
  xml:
    wrapped: true
```

```
<animals>
  <animals>value</animals>
  <animals>value</animals>
</animals>
```

To overcome the above example, the following definition can be used:

```
{
  "animals": {
    "type": "array",
    "items": {
      "type": "string",
      "xml": {
        "name": "animal"
      }
    },
    "xml": {
      "wrapped": true
    }
  }
}
```

```yaml
animals:
  type: array
  items:
    type: string
    xml:
      name: animal
  xml:
    wrapped: true
```

```xml
<animals>
  <animal>value</animal>
  <animal>value</animal>
</animals>
```

Affecting both internal and external names:

```json
{
  "animals": {
    "type": "array",
    "items": {
      "type": "string",
      "xml": {
        "name": "animal"
      }
    },
    "xml": {
      "name": "aliens",
      "wrapped": false
    }
    }
  }
```

```yaml
animals:
  type: array
  items:
    type: string
    xml:
      name: animal
  xml:
    name: aliens
    wrapped: false
```

```xml
<aliens>
  <animal>value</animal>
  <animal>value</animal>
</aliens>
```

If we change the external element but not the internal ones:

```json
{
  "animals": {
    "type": "array",
    "items": {
      "type": "string"
    },
    "xml": {
      "name": "aliens",
      "wrapped": true
    }
    }
  }
```

```yaml
animals:
  type: array
  items:
    type: string
  xml:
    name: aliens
    wrapped: true
```

```xml
<aliens>
  <aliens>value</aliens>
  <aliens>value</aliens>
</aliens>
```

## Definitions Object

An object to hold data types that can be consumed and produced by operations. These data types can be primitives, arrays or models.

#### Patterned Fields

| Field Pattern | Type | Description |
|---|---|---|
| {name} | Schema Object | A single definition, mapping a "name" to the schema it defines. |

#### Definitions Object Example

```json
{
  "Category": {
    "properties": {
      "id": {
        "type": "integer",
        "format": "int64"
      },
      "name": {
        "type": "string"
      }
    }
  },
  "Tag": {
    "properties": {
      "id": {
        "type": "integer",
        "format": "int64"
      },
      "name": {
        "type": "string"
      }
    }
  }
}
```

```yaml
Category:
  properties:
    id:
      type: integer
      format: int64
    name:
      type: string
  Tag:
    properties:
      id:
        type: integer
        format: int64
      name:
        type: string
```

## Parameters Definitions Object

An object to hold parameters to be reused across operations. Parameter definitions can be referenced to the ones defined here.

This does *not* define global operation parameters.

#### Patterned Fields

| Field Pattern | Type | Description |
|---|---|---|
| {name} | Parameter Object | A single parameter definition, mapping a "name" to the parameter it defines. |

#### Parameters Definition Object Example

```
{
  "skipParam": {
    "name": "skip",
    "in": "query",
    "description": "number of items to skip",
    "required": true,
    "type": "integer",
    "format": "int32"
    },
    "limitParam": {
      "name": "limit",
      "in": "query",
      "description": "max records to return",
      "required": true,
      "type": "integer",
      "format": "int32"
    }
  }
```

```
skipParam:
  name: skip
  in: query
  description: number of items to skip
  required: true
  type: integer
  format: int32
limitParam:
  name: limit
  in: query
  description: max records to return
  required: true
  type: integer
  format: int32
```

### Responses Definitions Object

An object to hold responses to be reused across operations. Response definitions can be referenced to the ones defined here.

This does *not* define global operation responses.

#### Patterned Fields

| Field Pattern | Type | Description |
|---|---|---|
| {name} | Response Object | A single response definition, mapping a "name" to the response it defines. |

#### Responses Definitions Object Example

```json
{
  "NotFound": {
    "description": "Entity not found."
  },
  "IllegalInput": {
    "description": "Illegal input for operation."
  },
  "GeneralError": {
    "description": "General Error",
    "schema": {
        "$ref": "#/definitions/GeneralError"
    }
  }
}
```

```yaml
NotFound:
  description: Entity not found.
IllegalInput:
  description: Illegal input for operation.
GeneralError:
  description: General Error
```

```
    schema:
      $ref: '#/definitions/GeneralError'
```

### Security Definitions Object

A declaration of the security schemes available to be used in the specification. This does not enforce the security schemes on the operations and only serves to provide the relevant details for each scheme.

#### Patterned Fields

| Field Pattern | Type | Description |
|---|---|---|
| {name} | Security Scheme Object | A single security scheme definition, mapping a "name" to the scheme it defines. |

#### Security Definitions Object Example

```
{
  "api_key": {
    "type": "apiKey",
    "name": "api_key",
    "in": "header"
  },
  "petstore_auth": {
    "type": "oauth2",
    "authorizationUrl": "http://swagger.io/api/oauth/dialog",
    "flow": "implicit",
    "scopes": {
      "write:pets": "modify pets in your account",
      "read:pets": "read your pets"
      }
    }
  }
```

```
api_key:
  type: apiKey
  name: api_key
  in: header
petstore_auth:
  type: oauth2
  authorizationUrl: http://swagger.io/api/oauth/dialog
  flow: implicit
  scopes:
    write:pets: modify pets in your account
    read:pets: read your pets
```

### Security Scheme Object

Allows the definition of a security scheme that can be used by the operations. Supported schemes are basic authentication, an API key (either as a header or as a query parameter) and OAuth2's common flows (implicit, password, application and access code).

#### Fixed Fields

| Field Name | Type | Validity | Description |
|---|---|---|---|
| type | `string` | Any | **Required.** The type of the security scheme. Valid values are `"basic"` , `"apiKey"` or `"oauth2"` . |
| description | `string` | Any | A short description for security scheme. |

| | | | |
|---|---|---|---|
| name | `string` | `apiKey` | **Required.** The name of the header or query parameter to be used. |
| in | `string` | `apiKey` | **Required** The location of the API key. Valid values are `"query"` or `"header"` . |
| flow | `string` | `oauth2` | **Required.** The flow used by the OAuth2 security scheme. Valid values are `"implicit"` , `"password"` , `"application"` or `"accessCode"` . |
| authorizationUrl | `string` | `oauth2` ( `"implicit"` , `"accessCode"` ) | **Required.** The authorization URL to be used for this flow. This SHOULD be in the form of a URL. |
| tokenUrl | `string` | `oauth2` ( `"password"` , `"application"` , `"accessCode"` ) | **Required.** The token URL to be used for this flow. This SHOULD be in the form of a URL. |
| scopes | Scopes Object | `oauth2` | **Required.** The available scopes for the OAuth2 security scheme. |

#### Patterned Fields

| Field Name | Type | Description |
|---|---|---|
| ^x- | Any | Allows extensions to the Swagger Schema. The field name MUST begin with `x-` , for example, `x-internal-id` . The value can be `null` , a primitive, an array or an object. See Vendor Extensions for further details. |

#### Security Scheme Object Example

#### Basic Authentication Sample

```
{
  "type": "basic"
}
```

```
type: basic
```

#### API Key Sample

```
{
  "type": "apiKey",
  "name": "api_key",
  "in": "header"
}
```

```
type: apiKey
name: api_key
in: header
```

#### Implicit OAuth2 Sample

```
{
  "type": "oauth2",
  "authorizationUrl": "http://swagger.io/api/oauth/dialog",
  "flow": "implicit",
  "scopes": {
    "write:pets": "modify pets in your account",
      "read:pets": "read your pets"
    }
  }
```

```
type: oauth2
authorizationUrl: http://swagger.io/api/oauth/dialog
flow: implicit
scopes:
  write:pets: modify pets in your account
  read:pets: read your pets
```

## Scopes Object

Lists the available scopes for an OAuth2 security scheme.

### Patterned Fields

| Field Pattern | Type | Description |
|---|---|---|
| {name} | `string` | Maps between a name of a scope to a short description of it (as the value of the property). |

### Scopes Object Example

```
{
  "write:pets": "modify pets in your account",
  "read:pets": "read your pets"
}
```

```
write:pets: modify pets in your account
read:pets: read your pets
```

## Security Requirement Object

Lists the required security schemes to execute this operation. The object can have multiple security schemes declared in it which are all required (that is, there is a logical AND between the schemes).

The name used for each property MUST correspond to a security scheme declared in the Security Definitions.

### Patterned Fields

| Field Pattern | Type | Description |
|---|---|---|
| {name} | [ `string` ] | Each name must correspond to a security scheme which is declared in the Security Definitions. If the security scheme is of type `"oauth2"` , then the value is a list of scope names required for the execution. For other security scheme types, the array MUST be empty. |

### Security Requirement Object Examples

#### Non-OAuth2 Security Requirement

```
{
  "api_key": []
}
```

```
api_key: []
```

#### OAuth2 Security Requirement

```
{
  "petstore_auth": [
    "write:pets",
    "read:pets"
  ]
}
```

```
petstore_auth:
- write:pets
- read:pets
```

## Specification Extensions

While the Swagger Specification tries to accommodate most use cases, additional data can be added to extend the specification at certain points.

The extensions properties are always prefixed by `"x-"` and can have any valid JSON format value.

The extensions may or may not be supported by the available tooling, but those may be extended as well to add requested support (if tools are internal or open-sourced).

## Security Filtering

Some objects in the Swagger specification may be declared and remain empty, or completely be removed, even though they are inherently the core of the API documentation.

The reasoning behind it is to allow an additional layer of access control over the documentation itself. While not part of the specification itself, certain libraries may choose to allow access to parts of the documentation based on some form of authentication/authorization.

Two examples for this: 1. The Paths Object may be empty. It may be counterintuitive, but this may tell the viewer that they got to the right place, but can't access any documentation. They'd still have access to the Info Object which may contain additional information regarding authentication. 2. The Path Item Object may be empty. In this case, the viewer will be aware that the path exists, but will not be able to see any of its operations or parameters. This is different than hiding the path itself from the Paths Object so the user will not be aware of its existence. This allows the documentation provider a finer control over what the viewer can see.