Swagger RESTful API Documentation Specification

Version 1.2 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.

1. 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 resulted files, for example, testing tools.

2. Revision History

Version | Date | Notes — | — | — 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

3. Definitions

- Resource: A `resource` in Swagger is a an entity that has a set of exposed operations. The entity can represent an actual object (pets, users..) or a set of logical operations collated together. It is up to the specification user to decide whether sub-resources should be referred to as part of their main resource or as a resource of their own. For example, assume the following URL set: "‘
- /users - GET POST
- /users/{id} - GET PATCH DELETE "‘ In this case, there's either one "/users" resource that contains operations on the "/users/{id}" sub-resource, or two separate resources.
- URL: A fully qualified URL.

4. Specification

4.1 Format

The files describing the RESTful API in accordance with the Swagger specification are represented as JSON objects and conform to the JSON standards.

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

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

Please note that while the API is described using JSON, it does not mean that the input nor output can't be in XML, YAML, plain text, or whichever format you chose to use with your API.

Unless noted otherwise, all field names in the specification are **case sensitive**.

4.2 File Structure

The Swagger representation of the API is comprised of two file types:

1. **The Resource Listing** - This is the root document that contains general API information and lists the resources. Each resource has its own URL that defines the API operations on it.
2. **The API Declaration** - This document describes a resource, including its API calls and models. There is one file per resource.

4.3 Data Types

In the Swagger specification, the data types are used in several locations - Operations, Operation Parameters, Models, and within the data types themselves (arrays).

The fields used to describe a given data type are added flatly to the relevant object. For example, if we have the object Foo with the field `name`, and we say it can also represent a data type, we MUST add the field `type` (or its variance, as explained ahead), and as such Foo would look like: `js "Foo" : { "name" : "sample", "type" : "string", ... }`

This section describes the general fields that are available to describe such data types. Some data types allow additional fields to extend further limitations on the data type *value* (see 4.3.3 Data Type Fields for further details).

Special care should be used when referencing a model (or a complex type). There are currently two variations, and the proper variation would be documented everywhere it may be used. This behavior will be unified in future versions of the spec.

The Swagger specification supports five data types:

1. `primitive` (input/output)
2. containers (as arrays/sets) (input/output)
3. complex (as `models`) (input/output)
4. `void` (output)
5. `File` (input)

4.3.1 Primitives Different programming languages represent primitives differently. The Swagger specification supports by name only the primitive types supported by the JSON-Schema Draft 4. However, in order to allow fine tuning a primitive definition, an additional `format` field MAY accompany the `type` primitive to give more information about the type used. If the `format` field is used, the respective client MUST conform to the elaborate type.

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` | dateTime | `string` | `date-time` |

4.3.2 `void` This value type is used to indicate that an operation returns no value. As such it MAY be used only for operations return type.

4.3.3 Data Type Fields As explained above, when an object is said to include a data type, there are a set of fields may be added to it (in fact, some are required, and some are optional).

Special care should be used when referencing a model (or a complex type). There currently two variations, and the proper variation would be documented everywhere it may be used. This behavior will be unified in future versions of the spec.

The table below shows the available fields to describe a data type. Note the `Validity` column may impose additional restrictions as to which data type is required in order to include this field. For example, `enum` may only be included of the `type` field is set to `string`.

Field Name | Type | Validity |Description —|:—:|—|— type | `string` | Any |**Required (if `$ref` is not used).** The return type of the operation. The value MUST be one of the Primitves, `array` or a model's `id`. $ref | `string` | Any | **Required (if `type` is not used).** The Model to be used. The value MUST be a model's `id`. format | `string` | primitive | Fine-tuned primitive type definition. See Primitives for further information. The value MUST be one that is defined under Primitives, corresponding to the right primitive `type`. defaultValue | *special* | primitive | The default value to be used for the field. The value type MUST conform with the primitive's `type` value. enum | [`string`] | `string` | A fixed list of possible values. If this field is used in conjunction with the `defaultValue` field, then the default value MUST be one of the values defined in the `enum`. minimum | `string` | `number`, `integer` | The minimum valid value for the type, inclusive. If this field is used in conjunction with the `defaultValue` field, then the default value MUST be higher than or equal to this value. The value type is `string` and should represent the minimum numeric value. **Note**: This will change to a numeric value in the future. maximum | `string` | `number`, `integer` | The maximum valid value for the type, inclusive. If this field is used in conjunction with the `defaultValue` field, then the default value MUST be lower than or equal to this value. The value type is `string` and should represent the maximum numeric value. **Note**: This will change to a numeric value in the future. items | Items Object | `array` | **Required.** The type definition of the values in the container. A container MUST NOT be nested in another container. uniqueItems | `boolean` | `array` | A flag to note whether the container allows duplicate values or not. If the value is set to `true`, then the `array` acts as a set.

4.3.4 Items Object This object is used to describe the value types used inside an array. Out of the Data Type Fields it can include either the `type` + `format` fields *OR* the `$ref` field (when referencing a model). The rest of the listed fields are not applicable.

In this case, `type` MUST NOT be `array`. Currently, there's no support for containers within containers.

4.3.4.1 Object Examples For a primitive type: `js { "type": "string" }`

For a complex type (model):

```
{
  "$ref": "Pet"
}
```

4.3.5 `File` The `File` (case sensitive) is a special type used to denote file upload. Note that declaring a model with the name `File` may lead to various conflicts

with third party tools and SHOULD be avoided.

When using `File`, the `consumes` field MUST be `"multipart/form-data"`, and the `paramType` MUST be `"form"`.

5. Schema

5.1 Resource Listing

The Resource Listing serves as the root document for the API description. It contains general information about the API and an inventory of the available resources.

By default, this document SHOULD be served at the `/api-docs` path.

Field Name | Type | Description —|:—:|— swaggerVersion | `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 an existing Swagger specification version. Currently, `"1.0"`, `"1.1"`, `"1.2"` are valid values. The field is of `string` value for possible non-numeric versions in the future (for example, "1.2a"). apis | [Resource Object] | **Required.** Lists the resources to be described by this specification implementation. The array can have 0 or more elements. apiVersion| `string` | Provides the version of the application API (not to be confused by the specification version). info | Info 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. authorizations | Authorizations Object | Provides information about the the authorization schemes allowed on his API.

5.1.1 Object Example

```
{
  "apiVersion": "1.0.0",
  "swaggerVersion": "1.2",
  "apis": [
    {
      "path": "/pet",
      "description": "Operations about pets"
    },
    {
      "path": "/user",
      "description": "Operations about user"
    },
    {
      "path": "/store",
      "description": "Operations about store"
```

```
    }
  ],
  "authorizations": {
    "oauth2": {
      "type": "oauth2",
      "scopes": [
        {
          "scope": "email",
          "description": "Access to your email address"
        },
        {
          "scope": "pets",
          "description": "Access to your pets"
        }
      ],
      "grantTypes": {
        "implicit": {
          "loginEndpoint": {
            "url": "http://petstore.swagger.wordnik.com/oauth/dialog"
          },
          "tokenName": "access_token"
        },
        "authorization_code": {
          "tokenRequestEndpoint": {
            "url": "http://petstore.swagger.wordnik.com/oauth/requestToken",
            "clientIdName": "client_id",
            "clientSecretName": "client_secret"
          },
          "tokenEndpoint": {
            "url": "http://petstore.swagger.wordnik.com/oauth/token",
            "tokenName": "access_code"
          }
        }
      }
    }
  },
  "info": {
    "title": "Swagger Sample App",
    "description": "This is a sample server Petstore server.  You can find out more about Sw
    "termsOfServiceUrl": "http://helloreverb.com/terms/",
    "contact": "apiteam@wordnik.com",
    "license": "Apache 2.0",
    "licenseUrl": "http://www.apache.org/licenses/LICENSE-2.0.html"
  }
}
```

5.1.2 Resource Object The Resource object describes a resource API endpoint in the application.

Field Name | Type | Description
—|:—:|— path | `string` | **Required.** A relative path to the API declaration from the path used to retrieve this Resource Listing. This `path` does not necessarily have to correspond to the URL which actually serves this resource in the API but rather where the resource listing itself is served. The value SHOULD be in a relative (URL) path format. description | `string` | *Recommended.* A short description of the resource.

5.1.2.1 Object Example:

```
{
    "path": "/pets",
    "description": "Operations about pets."
}
```

5.1.3 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.

Field Name | Type | Description
—|:—:|— title | `string` | **Required.** The title of the application.
description | `string` | **Required.** A short description of the application.
termsOfServiceUrl | `string` | A URL to the Terms of Service of the API. contact | `string` | An email to be used for API-related correspondence. license | `string` | The license name used for the API. licenseUrl | `string` | A URL to the license used for the API.

5.1.3.1 Object Example:

```
{
  "title": "Swagger Sample App",
  "description": "This is a sample server Petstore server.",
  "termsOfServiceUrl": "http://helloreverb.com/terms/",
  "contact": "apiteam@wordnik.com",
  "license": "Apache 2.0",
  "licenseUrl": "http://www.apache.org/licenses/LICENSE-2.0.html"
}
```

5.1.4 Authorizations Object The object provides information about the authorization schemes provided on this API. Currently, Swagger supports three

authorization schemes - basic authentication, API key and OAuth2. The Authorizations Object is used only to *declare* the available authorization schemes but not say which are required where. The actual authorization restrictions are done at the API declaration level.

Please note that the Authorizations Object is an object containing other object definitions and as such is structured as follows: `js { "Authorization1" : {...}, "Authorization2" : {...}, ..., "AuthorizationN" : {...} }`

Field Name | Type | Description —|:—:|— {Authorization Name} | Authorization Object | A new authorization definition. The name given to the {Authorization Name} is a friendly name that should be used when referring to the authorization scheme. In many cases, the {Authorization Name} used is the same as its type, but it can be anything.

5.1.4.1 Object Example:

```
{
  "oauth2": {
    "type": "oauth2",
    "scopes": [
      {
        "scope": "email",
        "description": "Access to your email address"
      },
      {
        "scope": "pets",
        "description": "Access to your pets"
      }
    ],
    "grantTypes": {
      "implicit": {
        "loginEndpoint": {
          "url": "http://petstore.swagger.wordnik.com/oauth/dialog"
        },
        "tokenName": "access_token"
      },
      "authorization_code": {
        "tokenRequestEndpoint": {
          "url": "http://petstore.swagger.wordnik.com/oauth/requestToken",
          "clientIdName": "client_id",
          "clientSecretName": "client_secret"
        },
        "tokenEndpoint": {
          "url": "http://petstore.swagger.wordnik.com/oauth/token",
          "tokenName": "access_code"
```

```
        }
      }
    }
  }
}
```

5.1.5 Authorization Object The object provides information about a specific authorization scheme. Currently, the authorization schemes supported are basic authentication, API key and OAuth2.

Within OAuth2, the Authorization Code Grant and Implicit Grant are supported.

In the table below, the `Validity` column imposes additional limitations to the requirement of the `type` in order to be able to use that field.

Field Name | Type | Validity | Description
—|:—:|—|— type | `string` | Any | **Required.** The type of the authorization scheme. Values MUST be either `"basicAuth"`, `"apiKey"` or `"oauth2"`.
passAs | `string` | `apiKey` | **Required.** Denotes how the API key must be passed. Valid values are `"header"` or `"query"`. keyname | `string` | `apiKey` | **Required.** The name of the `header` or `query` parameter to be used when passing the API key.
scopes | [Scope Object] | `oauth2` | A list of supported OAuth2 scopes. grantTypes | Grant Types Object | `oauth2` | **Required.** Detailed information about the grant types supported by the oauth2 authorization scheme.

5.1.5.1 Object Example:

```
  "oauth2": {
    "type": "oauth2",
    "scopes": [
      {
        "scope": "email",
        "description": "Access to your email address"
      },
      {
        "scope": "pets",
        "description": "Access to your pets"
      }
    ],
    "grantTypes": {
      "implicit": {
        "loginEndpoint": {
          "url": "http://petstore.swagger.wordnik.com/oauth/dialog"
        },
        "tokenName": "access_token"
```

```
      },
      "authorization_code": {
        "tokenRequestEndpoint": {
          "url": "http://petstore.swagger.wordnik.com/oauth/requestToken",
          "clientIdName": "client_id",
          "clientSecretName": "client_secret"
        },
        "tokenEndpoint": {
          "url": "http://petstore.swagger.wordnik.com/oauth/token",
          "tokenName": "access_code"
        }
      }
    }
  }
```

5.1.6 Scope Object Describes an OAuth2 authorization scope.

Field Name | Type | Description
—|:—:|— scope | `string` | **Required.** The name of the scope.
description | `string` | *Recommended.* A short description of the scope.

5.1.6.1 Object Example:

```
{
  "scope": "email",
  "description": "Access to your email address"
}
```

5.1.7 Grant Types Object Provides details regarding the OAuth2 grant types that are supported by the API. Currently, the Authorization Code and Implicit grants are supported.

At least one of the grant types MUST be included (otherwise there's no need for the OAuth2 declaration).

Field Name | Type | Description
—|:—:|— implicit | Implicit Object | The Implicit Grant flow definition.
authorization_code | Authorization Code Object | The Authorization Code Grant flow definition.

5.1.7.1 Object Example:

```
{
  "implicit": {
```

```
    "loginEndpoint": {
      "url": "http://petstore.swagger.wordnik.com/oauth/dialog"
    },
    "tokenName": "access_token"
  },
  "authorization_code": {
    "tokenRequestEndpoint": {
      "url": "http://petstore.swagger.wordnik.com/oauth/requestToken",
      "clientIdName": "client_id",
      "clientSecretName": "client_secret"
    },
    "tokenEndpoint": {
      "url": "http://petstore.swagger.wordnik.com/oauth/token",
      "tokenName": "access_code"
    }
  }
}
```

5.1.8 Implicit Object Provides details regarding the OAuth2's Implicit Grant flow type.

Field Name | Type | Description
—|:—:|— loginEndpoint | Login Endpoint Object | **Required.** The login endpoint definition.
tokenName | `string` | An optional alternative name to standard "access_token" OAuth2 parameter.

5.1.8.1 Object Example:

```
{
  "loginEndpoint": {
    "url": "http://petstore.swagger.wordnik.com/oauth/dialog"
  },
  "tokenName": "access_token"
}
```

5.1.9 Authorization Code Object Provides details regarding the OAuth2's Authorization Code Grant flow type.

Field Name | Type | Description
—|:—:|— tokenRequestEndpoint | Token Request Endpoint Object | **Required.** The token request endpoint definition. tokenEndpoint | Token Endpoint Object | **Required.** The token endpoint definition.

5.1.9.1 Object Example:

```
{
  "tokenRequestEndpoint": {
    "url": "http://petstore.swagger.wordnik.com/oauth/requestToken",
    "clientIdName": "client_id",
    "clientSecretName": "client_secret"
  },
  "tokenEndpoint": {
    "url": "http://petstore.swagger.wordnik.com/oauth/token",
    "tokenName": "access_code"
  }
}
```

5.1.10 Login Endpoint Object Provides details regarding the Implicit Grant's *authorization endpoint*.

Field Name | Type | Description
—|:—:|— url | `string` | **Required.** The URL of the authorization endpoint for the implicit grant flow. The value SHOULD be in a URL format.

5.1.10.1 Object Example:

```
{
  "url": "http://petstore.swagger.wordnik.com/oauth/dialog"
}
```

5.1.11 Token Request Endpoint Object Provides details regarding the OAuth2's *Authorization Endpoint*.

Field Name | Type | Description
—|:—:|— url | `string` | **Required.** The URL of the authorization endpoint for the authentication code grant flow. The value SHOULD be in a URL format. clientIdName | `string` | An optional alternative name to standard "client_id" OAuth2 parameter. clientSecretName | `string` | An optional alternative name to standard "client_secret" OAuth2 parameter.

5.1.11.1 Object Example:

```
{
  "url": "http://petstore.swagger.wordnik.com/oauth/requestToken",
  "clientIdName": "client_id",
  "clientSecretName": "client_secret"
}
```

5.1.12 Token Endpoint Object Provides details regarding the OAuth2's *Token Endpoint.*

Field Name | Type | Description
—|:—:|— url | `string` | **Required.** The URL of the token endpoint for the authentication code grant flow. The value SHOULD be in a URL format. tokenName | `string` | An optional alternative name to standard "access_token" OAuth2 parameter.

5.1.12.1 Object Example:

```
{
  "url": "http://petstore.swagger.wordnik.com/oauth/token",
  "tokenName": "access_code"
}
```

5.2 API Declaration

The API Declaration provides information about an API exposed on a resource. There should be one file per Resource described. The file MUST be served in the URL described by the `path` field.

Field Name | Type | Description —|:—:|— swaggerVersion | `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 an existing Swagger specification version. Currently, `"1.0"`, `"1.1"`, `"1.2"` are valid values. apiVersion | `string` | Provides the version of the application API (not to be confused by the specification version). basePath | `string` | **Required.** The root URL serving the API. This field is important as while it is common to have the Resource Listing and API Declarations on the server providing the APIs themselves, it is not a requirement. The API specifications can be served using static files and not generated by the API server itself, so the URL for serving the API cannot always be derived from the URL serving the API specification. The value SHOULD be in the format of a URL. resourcePath | `string` | The *relative* path to the resource, from the `basePath`, which this API Specification describes. The value MUST precede with a forward slash (`"/"`). apis | [API Object] | **Required.** A list of the APIs exposed on this resource. There MUST NOT be more than one API Object per `path` in the array. models | Models Object | A list of the models available to this resource. Note that these need to be exposed separately for each API Declaration. produces | [`string`] | A list of MIME types the APIs on this resource can produce. This is global to all APIs but can be overridden on specific API calls. consumes | [`string`] | A list of MIME types the APIs on this resource can consume. This is global to all APIs but can be overridden on specific API calls. authorizations | Authorizations Object | A list of authorizations schemes *required* for the operations listed in

this API declaration. Individual operations may override this setting. If there are multiple authorization schemes described here, it means they're **all** applied.

5.2.1 Object Example


```
{
  "apiVersion": "1.0.0",
  "swaggerVersion": "1.2",
  "basePath": "http://petstore.swagger.wordnik.com/api",
  "resourcePath": "/store",
  "produces": [
    "application/json"
  ],
  "authorizations": {},
  "apis": [
    {
      "path": "/store/order/{orderId}",
      "operations": [
        {
          "method": "GET",
          "summary": "Find purchase order by ID",
          "notes": "For valid response try integer IDs with value <= 5. Anything above 5 or
          "type": "Order",
          "nickname": "getOrderById",
          "authorizations": {},
          "parameters": [
            {
              "name": "orderId",
              "description": "ID of pet that needs to be fetched",
              "required": true,
              "type": "string",
              "paramType": "path"
            }
          ],
          "responseMessages": [
            {
              "code": 400,
              "message": "Invalid ID supplied"
            },
            {
              "code": 404,
              "message": "Order not found"
            }
          ]
        },
```

```
    {
      "method": "DELETE",
      "summary": "Delete purchase order by ID",
      "notes": "For valid response try integer IDs with value < 1000.  Anything above 1
      "type": "void",
      "nickname": "deleteOrder",
      "authorizations": {
        "oauth2": [
          {
            "scope": "test:anything",
            "description": "anything"
          }
        ]
      },
      "parameters": [
        {
          "name": "orderId",
          "description": "ID of the order that needs to be deleted",
          "required": true,
          "type": "string",
          "paramType": "path"
        }
      ],
      "responseMessages": [
        {
          "code": 400,
          "message": "Invalid ID supplied"
        },
        {
          "code": 404,
          "message": "Order not found"
        }
      ]
    }
  ]
},
{
  "path": "/store/order",
  "operations": [
    {
      "method": "POST",
      "summary": "Place an order for a pet",
      "notes": "",
      "type": "void",
      "nickname": "placeOrder",
      "authorizations": {
```

```
          "oauth2": [
            {
              "scope": "test:anything",
              "description": "anything"
            }
          ]
        },
        "parameters": [
          {
            "name": "body",
            "description": "order placed for purchasing the pet",
            "required": true,
            "type": "Order",
            "paramType": "body"
          }
        ],
        "responseMessages": [
          {
            "code": 400,
            "message": "Invalid order"
          }
        ]
      }
    ]
  }
],
"models": {
  "Order": {
    "id": "Order",
    "properties": {
      "id": {
        "type": "integer",
        "format": "int64"
      },
      "petId": {
        "type": "integer",
        "format": "int64"
      },
      "quantity": {
        "type": "integer",
        "format": "int32"
      },
      "status": {
        "type": "string",
        "description": "Order Status",
        "enum": [
```

```
            "placed",
            " approved",
            " delivered"
          ]
        },
        "shipDate": {
          "type": "string",
          "format": "date-time"
        }
      }
    }
  }
}
```

5.2.2 API Object The API Object describes one or more operations on a single `path`. In the `apis` array, there MUST be only one `API Object` per `path`.

Field Name | Type | Description —|:—:|— path | `string` | **Required.** The relative path to the operation, from the `basePath`, which this operation describes. The value SHOULD be in a relative (URL) path format. description | `string` | *Recommended.* A short description of the resource. operations | [Operation Object] | **Required.** A list of the API operations available on this path. The array may include 0 or more operations. There MUST NOT be more than one Operation Object per `method` in the array.

5.2.2.1 Object Example:

```
{
"path": "/pet",
"operations": [
  {
    "method": "PUT",
    "summary": "Update an existing pet",
    "notes": "",
    "type": "void",
    "nickname": "updatePet",
    "authorizations": {},
    "parameters": [
      {
        "name": "body",
        "description": "Pet object that needs to be updated in the store",
        "required": true,
        "type": "Pet",
        "paramType": "body"
      }
```

```
    ],
    "responseMessages": [
      {
        "code": 400,
        "message": "Invalid ID supplied"
      },
      {
        "code": 404,
        "message": "Pet not found"
      },
      {
        "code": 405,
        "message": "Validation exception"
      }
    ]
  },
  {
    "method": "POST",
    "summary": "Add a new pet to the store",
    "notes": "",
    "type": "void",
    "nickname": "addPet",
    "consumes": [
      "application/json",
      "application/xml"
    ],
    "authorizations": {
      "oauth2": [
        {
          "scope": "test:anything",
          "description": "anything"
        }
      ]
    },
    "parameters": [
      {
        "name": "body",
        "description": "Pet object that needs to be added to the store",
        "required": true,
        "type": "Pet",
        "paramType": "body"
      }
    ],
    "responseMessages": [
      {
        "code": 405,
```

```
            "message": "Invalid input"
          }
        ]
      }
    ]
  }
```

5.2.3 Operation Object The Operation Object describes a single operation on a `path`.

In the `operations` array, there MUST be only one `Operation Object` per `method`.

This object includes the Data Type Fields in order to describe the return value of the operation. The `type` field MUST be used to link to other models.

This is the only object where the `type` MAY have the value of `void` to indicate that the operation returns no value.

Field Name | Type | Description —|:—:|— method | `string` | **Required.** The HTTP method required to invoke this operation. The value MUST be one of the following values: `"GET"`, `"HEAD"`, `"POST"`, `"PUT"`, `"PATCH"`, `"DELETE"`, `"OPTIONS"`. Note that the values MUST be in uppercase. 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. notes | `string` | A verbose explanation of the operation behavior. nickname |`string` | **Required.** A unique id for the operation that can be used by tools reading the output for further and easier manipulation. For example, Swagger-Codegen will use the nickname as the method name of the operation in the client it generates. The value MUST be alphanumeric and may include underscores. Whitespsace characters are not allowd. authorizations | Authorizations Object | A list of authorizations required to execute this operation. While not mandatory, if used, it overrides the value given at the API Declaration's authorizations. In order to completely remove API Declaration's authorizations completely, an empty object (`{}`) may be applied. parameters | [Parameter Object] | **Required.** The inputs to the operation. If no parameters are needed, an empty array MUST be included. responseMessages | [Response Message Object] | Lists the possible response statuses that can return from the operation. produces | [`string`] | A list of MIME types this operation can produce. This is overrides the global `produces` definition at the root of the API Declaration. Each `string` value SHOULD represent a MIME type. consumes | [`string`] | A list of MIME types this operation can consume. This is overrides the global `consumes` definition at the root of the API Declaration. Each `string` value SHOULD represent a MIME type. deprecated | `string` | Declares this operation to be deprecated. Usage of the declared operation should be refrained. Valid value MUST be either `"true"` or `"false"`. *Note:* This field will change to type `boolean` in the future.

5.2.3.1 Object Example

```
{
  "method": "GET",
  "summary": "Find pet by ID",
  "notes": "Returns a pet based on ID",
  "type": "Pet",
  "nickname": "getPetById",
  "authorizations": {},
  "parameters": [
    {
      "name": "petId",
      "description": "ID of pet that needs to be fetched",
      "required": true,
      "type": "integer",
      "format": "int64",
      "paramType": "path",
      "minimum": "1.0",
      "maximum": "100000.0"
    }
  ],
  "responseMessages": [
    {
      "code": 400,
      "message": "Invalid ID supplied"
    },
    {
      "code": 404,
      "message": "Pet not found"
    }
  ]
}
```

5.2.4 Parameter Object The Parameter Object describes a single parameter to be sent in an operation and maps to the `parameters` field in the Operation Object.

This object includes the Data Type Fields in order to describe the type of this parameter. The `type` field MUST be used to link to other models.

If `type` is `File`, the `consumes` field MUST be `"multipart/form-data"`, and the `paramType` MUST be `"form"`.

Field Name | Type | Description —|:—:|— paramType | `string` | **Required.** The type of the parameter (that is, the location of the parameter in the request). The value MUST be one of these values: `"path"`, `"query"`, `"body"`, `"header"`, `"form"`. Note that the values MUST be lower case. name | `string` | **Required.**

The unique name for the parameter. Each `name` MUST be unique, even if they are associated with different `paramType` values. Parameter names are *case sensitive*.

If `paramType` is `"path"`, the `name` field MUST correspond to the associated path segment from the `path` field in the API Object.

If `paramType` is `"query"`, the `name` field corresponds to the query parameter name.

If `paramType` is `"body"`, the name is used only for Swagger-UI and Swagger-Codegen. In this case, the `name` MUST be `"body"`.

If `paramType` is `"form"`, the `name` field corresponds to the form parameter key.

If `paramType` is `"header"`, the `name` field corresponds to the header parameter key.

See here for some examples. description | `string` | *Recommended.* A brief description of this parameter. required | `boolean` | A flag to note whether this parameter is required. If this field is not included, it is equivalent to adding this field with the value `false`. The field MUST be included if `paramType` is `"path"` and MUST have the value `true`. allowMultiple | `boolean` | Another way to allow multiple values for a "query" parameter. If used, the query parameter may accept comma-separated values. The field may be used only if `paramType` is `"query"`, `"header"` or `"path"`.

5.2.4.1 Name Examples

- If `paramType` is `"path"`, and assuming the `path` is `"/pet/{id}"`:

  `"name": "id"`

- If `paramType` is `"query"`, and assuming the URL call is `"http://host/resource?limit=100"` (that is, there's a query parameter called `"limit"`:

  `"name": "limit"`

- If `paramType` is `"body"`:

  `"name": "body"`

5.2.4.2 Object Example

```
{
  "name": "body",
  "description": "Pet object that needs to be updated in the store",
```

```
  "required": true,
  "type": "Pet",
  "paramType": "body"
}
```

5.2.5 Response Message Object The Response Message Object describes a single possible response message that can be returned from the operation call, and maps to the `responseMessages` field in the Operation Object. Each Response Message allows you to give further details as to why the HTTP status code may be received.

Field Name | Type | Description | —|:—:|— code | `integer` | **Required.** The HTTP status code returned. The value SHOULD be one of the status codes as described in RFC 2616 - Section 10. message | `string` | **Required.** The explanation for the status code. It SHOULD be the reason an error is received if an error status code is used. responseModel | `string` | The return type for the given response.

5.2.5.1 Object Example

```
  {
    "code": 404,
    "message": "no project found",
    "responseModel": "ErrorModel"
  }
```

5.2.6 Models Object The Models Object holds a field per model definition, and this is different than the structure of the other objects in the spec. It follows a subset of the JSON-Schema specification.

Please note that the Models Object is an object containing other object definitions and as such is structured as follows: `js { "Model1" : {...}, "Model2" : {...}, ..., "ModelN" : {...} }`

Field Name | Type | Description —|:—:|— {Model Name} | Model Object | A new model definition. Note the actual name of the field is the name you're giving your model. For example, "Category", "Pet", "User".

5.2.6.1 Object Example

```
{
    "Category": {
      "id": "Category",
      "properties": {
```

```
        "id": {
          "type": "integer",
          "format": "int64"
        },
        "name": {
          "type": "string"
        }
      }
    }
}
```

5.2.7 Model Object A Model Object holds the definition of a new model for this API Declaration.

Models in Swagger allow for inheritance. The inheritance is controlled by two fields - `subTypes` to give the name of the models extending this definition, and `discriminator` to support polymorphism.

Field Name | Type | Description
—|:—:|— id | `string` | **Required.** A unique identifier for the model. This MUST be the name given to {Model Name}. description | `string` | A brief description of this model. required | [`string`] | A definition of which properties MUST exist when a model instance is produced. The values MUST be the `{Property Name}` of one of the `properties`. properties | Properties Object | **Required.** A list of properties (fields) that are part of the model subTypes | [`string`] | List of the model `id`s that inherit from this model. Sub models inherit all the properties of the parent model. Since inheritance is transitive, if the parent of a model inherits from another model, its sub-model will include all properties. As such, if you have `Foo->Bar->Baz`, then Baz will inherit the properties of Bar and Foo. There MUST NOT be a cyclic definition of inheritance. For example, if `Foo -> ... -> Bar`, having `Bar -> ... -> Foo` is not allowed. There also MUST NOT be a case of multiple inheritance. For example, `Foo -> Baz <- Bar` is not allowed. A sub-model definition MUST NOT override the `properties` of any of its ancestors. All sub-models MUST be defined in the same API Declaration. discriminator | `string` | MUST be included only if `subTypes` is included. This field allows for polymorphism within the described inherited models. This field MAY be included at any base model but MUST NOT be included in a sub-model. The value of this field MUST be a name of one of the `properties` in this model, and that field MUST be in the `required` list. When used, the value of the *discriminator property* MUST be the name of parent or any of its sub-models (to any depth of inheritance).

5.2.7.1 Object Example

```
{
```

```
  "id": "Order",
  "properties": {
    "id": {
      "type": "integer",
      "format": "int64"
    },
    "petId": {
      "type": "integer",
      "format": "int64"
    },
    "quantity": {
      "type": "integer",
      "format": "int32"
    },
    "status": {
      "type": "string",
      "description": "OrderStatus",
      "enum": [
        "placed",
        "approved",
        "delivered"
      ]
    },
    "shipDate": {
      "type": "string",
      "format": "date-time"
    }
  }
}
```

5.2.7.2 Inheritance Example Say we have a general Animal model, and a sub-model for Cat.

```
"Animal": {
  "id": "Animal",
  "name": "Animal",
  "required": [
    "id",
    "type"
  ],
  "properties": {
    "id": {
      "type": "long"
    },
    "type": {
```

```
      "type": "string"
    }
  },
  "subTypes": ["Cat"],
  "discriminator": "type"
},
"Cat": {
  "id": "Cat",
  "name": "Cat",
  "required": [
    "likesMilk"
  ],
  "properties": {
    "likesMilk": {
      "type": "boolean"
    }
  },
}
```

5.2.8 Properties Object The Properties Object holds a field per property definition, and this is different than the structure of the other objects in the spec. It follows a subset of the JSON-Schema specification.

Please note that the Properties Object is an object containing other object definitions and as such is structured as follows: `js { "Property1" : {...}, "Property2" : {...}, ..., "PropertyN" : {...} }`

Field Name | Type | Description —|:—:|— {Property Name} | Property Object | A new model property definition. Note the actual name of the field is the name you're giving your property. For example, "id", "name", "age".

5.2.8.1 Object Example

```
{
  "id": {
    "type": "integer",
    "format": "int64"
  },
  "name": {
    "type": "string"
  }
}
```

5.2.9 Property Object A Property Object holds the definition of a new property for a model.

This object includes the Data Type Fields in order to describe the type of this property. The `$ref` field MUST be used to link to other models.

Properties MUST NOT contain other properties. If there's a need for an internal object hierarchy, additional models MUST be created and linked to a flat structure.

Field Name | Type | Description —|:—:|— description | `string` | *Recommended.* A brief description of this property.

5.2.9.1 Object Examples A simple 64bit integer field called "id", with a description and min/max values:

```
"id": {
    "type": "integer",
    "format": "int64",
    "description": "unique identifier for the pet",
    "minimum": "0.0",
    "maximum": "100.0"
}
```

A "category" field of a Category model.

```
"category": {
    "$ref": "Category"
}
```

A "tags" field of type array containing Tag models.

```
"tags": {
    "type": "array",
    "items": {
        "$ref": "Tag"
    }
}
```

5.2.10 Authorizations Object The object provides information about the authorization schemes enforced on this API. If used in the API Declaration's authorizations, it applies to all operations listed. If used in the Operation's authorizations, it applies to the operation itself and may override the API Declaration's authorizations. If multiple authorization schemes are described, they are **all** required to perform the operations listed.

Please note that the Authorizations Object is an object containing arrays of object definitions and as such is structured as follows: `js { "Authorization1" :`

```
[...], "Authorization2" :  [...], ..., "AuthorizationN" :  [...]
}
```

Field Name | Type | Description —|:—:|— {Authorization Name} | * | The authorization scheme to be used. The name given to the {Authorization Name} MUST be a friendly name that was given to an authorization scheme in the Resource Listing's authorizations. If the friendly name describes an OAuth2 security scheme, the value should be of type [Scope Object] (but may be an empty array to denote 'no scopes'). For all other authorization scheme types, the value MUST be an empty array.

5.2.10 Object Example:

```
{
  "oauth2": [
    {
      "scope": "write:pets",
      "description": "modify pets in your account"
    },
    {
      "scope": "read:pets",
      "description": "read your pets"
    }
  ]
}
```

5.2.11 Scope Object Describes an OAuth2 authorization scope. The scope described here MUST be described in the respective friendly name definition of the security scheme in the Resource Listing's authorizations.

Field Name | Type | Description
—|:—:|— scope | `string` | **Required.** The name of the scope.
description | `string` | *Recommended.* A short description of the scope.

5.2.11.1 Object Example:

```
{
  "scope": "email",
  "description": "Access to your email address"
}
```