

OpenAPI Specification

Version 3.0.2

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [BCP 14 RFC2119 RFC8174](#) when, and only when, they appear in all capitals, as shown here.

This document is licensed under [The Apache License, Version 2.0](#).

Introduction

The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to RESTful APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. When properly defined, a consumer can understand and interact with the remote service with a minimal amount of implementation logic.

An OpenAPI definition can then be used by documentation generation tools to display the API, code generation tools to generate servers and clients in various programming languages, testing tools, and many other use cases.

Table of Contents

- [Definitions](#)
 - [OpenAPI Document](#)
 - [Path Templating](#)
 - [Media Types](#)
 - [HTTP Status Codes](#)
- [Specification](#)
 - [Versions](#)
 - [Format](#)
 - [Document Structure](#)
 - [Data Types](#)
 - [Rich Text Formatting](#)
 - [Relative References In URLs](#)
 - [Schema](#)

- [OpenAPI Object](#)
- [Info Object](#)
- [Contact Object](#)
- [License Object](#)
- [Server Object](#)
- [Server Variable Object](#)
- [Components Object](#)
- [Paths Object](#)
- [Path Item Object](#)
- [Operation Object](#)
- [External Documentation Object](#)
- [Parameter Object](#)
- [Request Body Object](#)
- [Media Type Object](#)
- [Encoding Object](#)
- [Responses Object](#)
- [Response Object](#)
- [Callback Object](#)
- [Example Object](#)
- [Link Object](#)
- [Header Object](#)
- [Tag Object](#)
- [Reference Object](#)
- [Schema Object](#)
- [Discriminator Object](#)
- [XML Object](#)
- [Security Scheme Object](#)
- [OAuth Flows Object](#)
- [OAuth Flow Object](#)
- [Security Requirement Object](#)
- [Specification Extensions](#)
- [Security Filtering](#)
- [Appendix A: Revision History](#)

Definitions

A document (or set of documents) that defines or describes an API. An OpenAPI definition uses and conforms to the OpenAPI Specification.

Path Templating

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

Media Types

Media type definitions are spread across several resources. The media type definitions SHOULD be in compliance with [RFC6838](#).

Some examples of possible media 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 defined by [RFC7231](#) and registered status codes are listed in the [IANA Status Code Registry](#).

Specification

Versions

The OpenAPI Specification is versioned using [Semantic Versioning 2.0.0](#) (semver) and follows the semver specification.

The major.minor portion of the semver (for example 3.0) SHALL designate the OAS feature set. Typically, *.patch* versions address errors in this document, not the feature set. Tooling which supports OAS 3.0 SHOULD be compatible with all OAS 3.0.* versions. The patch version SHOULD NOT be considered by tooling, making no distinction between 3.0.0 and 3.0.1 for example.

Subsequent minor version releases of the OpenAPI Specification (incrementing the minor version number) SHOULD NOT interfere with tooling developed to a lower

minor version and same major version. Thus a hypothetical 3.1.0 specification SHOULD be usable with tooling designed for 3.0.0.

An OpenAPI document compatible with OAS 3.*.* contains a required [openapi](#) field which designates the semantic version of the OAS that it uses. (OAS 2.0 documents contain a top-level version field named [swagger](#) and value "2.0".)

Format

An OpenAPI document that conforms to the OpenAPI Specification is itself a JSON object, which may be represented either in JSON or YAML format.

For example, if a field has an array value, the JSON array representation will be used:

```
{
  "field": [ 1, 2, 3 ]
}
```

All field names in the specification are **case sensitive**. This includes all fields that are used as keys in a map, except where explicitly noted that keys are **case insensitive**.

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 MUST have unique names within the containing object.

In order to preserve the ability to round-trip between YAML and JSON formats, YAML version [1.2](#) is RECOMMENDED along with some additional constraints:

- Tags MUST be limited to those allowed by the [JSON Schema ruleset](#).
- Keys used in YAML maps MUST be limited to a scalar string, as defined by the [YAML Failsafe schema ruleset](#).

Note: While APIs may be defined by OpenAPI documents in either YAML or JSON format, the API request and response bodies and other content are not required to be JSON or YAML.

Document Structure

An OpenAPI document MAY be made up of a single document or be divided into multiple, connected parts at the discretion of the user. In the latter case, `$ref` fields MUST be used in the specification to reference those parts as follows from the [JSON Schema](#) definitions.

It is RECOMMENDED that the root OpenAPI document be named: `openapi.json` OR `openapi.yaml`.

Data Types

Primitive data types in the OAS are based on the types supported by the [JSON Schema Specification Wright Draft 00](#). Note that integer as a type is also supported and is defined as a JSON number without a fraction or exponent part. `null` is not supported as a type (see [nullable](#) for an alternative solution). Models are defined using the [Schema Object](#), which is an extended subset of JSON Schema Specification Wright Draft 00.

Primitives have an optional modifier property: `format`. OAS uses several known formats to define in fine detail the data type being used. However, to support documentation needs, the `format` property is an open string-valued property, and can have any value. Formats such as "email", "uuid", and so on, MAY be used even though undefined by this specification. Types that are not accompanied by a `format` property follow the type definition in the JSON Schema. Tools that do not recognize a specific format MAY default back to the type alone, as if the `format` is not specified.

The formats defined by the OAS are:

type	format	Comments
integer	int32	signed 32 bits
integer	int64	signed 64 bits (a.k.a long)
number	float	
number	double	
string		
string	byte	base64 encoded characters
string	binary	any sequence of octets
boolean		
string	date	As defined by full-date - RFC3339
string	date-time	As defined by date-time - RFC3339
string	password	A hint to UIs to obscure input.

Rich Text Formatting

Throughout the specification description fields are noted as supporting CommonMark markdown formatting. Where OpenAPI tooling renders rich text it MUST support, at a minimum, markdown syntax as described by [CommonMark 0.27](#). Tooling MAY choose to ignore some CommonMark features to address security concerns.

Relative References in URLs

Unless specified otherwise, all properties that are URLs MAY be relative references as defined by [RFC3986](#). Relative references are resolved using the URLs defined in the [Server Object](#) as a Base URI.

Relative references used in `$ref` are processed as per [JSON Reference](#), using the URL of the current document as the base URI. See also the [Reference Object](#).

Schema

In the following description, if a field is not explicitly **REQUIRED** or described with a MUST or SHALL, it can be considered OPTIONAL.

OpenAPI Object

This is the root document object of the [OpenAPI document](#).

Fixed Fields

Field Name	Type	Description
openapi	string	REQUIRED. This string MUST be the semantic version number of the OpenAPI Specification version that the OpenAPI document uses. The <code>openapi</code> field SHOULD be used by tooling specifications and clients to interpret the OpenAPI document. This is <i>not</i> related to the API <code>info.version</code> string.
info	Info Object	REQUIRED. Provides metadata about the API. The metadata MAY be used by tooling as required.
servers	[Server Object]	An array of Server Objects, which provide connectivity information to a target server. If the <code>servers</code> property is not provided, or is an empty array, the default value would be a Server Object with a <code>url</code> value of <code>/</code> .
paths	Paths Object	REQUIRED. The available paths and operations for the API.

Field Name	Type	Description
components	Components Object	An element to hold various schemas for the specification.
security	[Security Requirement Object]	A declaration of which security mechanisms can be used across the API. The list of values includes alternative security requirement objects that can be used. Only one of the security requirement objects need to be satisfied to authorize a request. 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 Object	Additional external documentation.

This object MAY be extended with [Specification Extensions](#).

Info Object

The object provides metadata about the API. The metadata MAY be used by the clients if needed, and MAY be presented in editing or documentation generation tools for convenience.

Fixed Fields

Field Name	Type	Description
title	string	REQUIRED. The title of the application.
description	string	A short description of the application. CommonMark syntax MAY be used for rich text representation.

Field Name	Type	Description
termsOfService	string	A URL to the Terms of Service for the API. MUST be in the format of a URL.
contact	Contact Object	The contact information for the exposed API.
license	License Object	The license information for the exposed API.
version	string	REQUIRED. The version of the OpenAPI document (which is distinct from the OpenAPI Specification version or the API implementation version).

This object MAY be extended with [Specification Extensions](#).

Info Object Example

```
{
  "title": "Sample Pet Store App",
  "description": "This is a sample server for a pet store.",
  "termsOfService": "http://example.com/terms/",
  "contact": {
    "name": "API Support",
    "url": "http://www.example.com/support",
    "email": "support@example.com"
  },
  "license": {
    "name": "Apache 2.0",
    "url": "https://www.apache.org/licenses/LICENSE-2.0.html"
  },
  "version": "1.0.1"
}
```

title: Sample Pet Store App
description: This is a sample server for a pet store.
termsOfService: http://example.com/terms/
contact:
 name: API Support
 url: http://www.example.com/support
 email: support@example.com
license:
 name: Apache 2.0
 url: https://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.

This object MAY be extended with [Specification Extensions](#).

Contact Object Example

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

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.

This object MAY be extended with specifications extensions

License Object Example

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

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

Server Object

An object representing a Server.

Fixed Fields

Field Name	Type	Description
url	string	REQUIRED. A URL to the target host. This URL supports Server Variables and MAY be relative, to indicate that the host location is relative to the location where the OpenAPI document is being served. Variable substitutions will be made when a variable is named in {brackets}.
description	string	An optional string describing the host designated by the URL. CommonMark syntax MAY be used for rich text representation.
variables	Map[string, Server Variable Object]	A map between a variable name and its value. The value is used for substitution in the server's URL template.

This object MAY be extended with [Specification Extensions](#).

Server Object Example

A single server would be described as:

```
{  
  "url": "https://development.gigantic-server.com/v1",  
  "description": "Development server"  
}  
url: https://development.gigantic-server.com/v1  
description: Development server
```

The following shows how multiple servers can be described, for example, at the OpenAPI Object's [servers](#):

```
{
  "servers": [
    {
      "url": "https://development.gigantic-server.com/v1",
      "description": "Development server"
    },
    {
      "url": "https://staging.gigantic-server.com/v1",
      "description": "Staging server"
    },
    {
      "url": "https://api.gigantic-server.com/v1",
      "description": "Production server"
    }
  ]
}
```

```
servers:
- url: https://development.gigantic-server.com/v1
  description: Development server
- url: https://staging.gigantic-server.com/v1
  description: Staging server
- url: https://api.gigantic-server.com/v1
  description: Production server
```

The following shows how variables can be used for a server configuration:

```
{
  "servers": [
    {
      "url": "https://{username}.gigantic-server.com:{port}/{basePath}",
      "description": "The production API server",
      "variables": {
        "username": {
          "default": "demo",
          "description": "this value is assigned by the service provider, in this
example `gigantic-server.com`"
        },
        "port": {
          "enum": [
            "8443",
            "443"
          ],
          "default": "8443"
        },
        "basePath": {
          "default": "v2"
        }
      }
    }
  ]
}
```

```
servers:
- url: https://{username}.gigantic-server.com:{port}/{basePath}
  description: The production API server
  variables:
    username:
      # note! no enum here means it is an open value
```

```

    default: demo
    description: this value is assigned by the service provider, in this example
`gigantic-server.com`
    port:
      enum:
        - '8443'
        - '443'
      default: '8443'
    basePath:
      # open meaning there is the opportunity to use special base paths as
      assigned by the provider, default is `v2`
      default: v2

```

Server Variable Object

An object representing a Server Variable for server URL template substitution.

Fixed Fields

Field Name	Type	Description
enum	[string]	An enumeration of string values to be used if the substitution options are from a limited set.
default	string	REQUIRED. The default value to use for substitution, which SHALL be sent if an alternate value is <i>not</i> supplied. Note this behavior is different than the Schema Object's treatment of default values, because in those cases parameter values are optional.
description	string	An optional description for the server variable. CommonMark syntax MAY be used for rich text representation.

This object MAY be extended with [Specification Extensions](#).

Components Object

Holds a set of reusable objects for different aspects of the OAS. All objects defined within the components object will have no effect on the API unless they are explicitly referenced from properties outside the components object.

Fixed Fields

Field Name	Type	Description
schemas	Map[string, Schema Object Reference Object]	An object to hold reusable Schema Objects .
responses	Map[string, Response Object Reference Object]	An object to hold reusable Response Objects .
parameters	Map[string, Parameter Object Reference Object]	An object to hold reusable Parameter Objects .
examples	Map[string, Example Object Reference Object]	An object to hold reusable Example Objects .
requestBodies	Map[string, Request Body Object Reference Object]	An object to hold reusable Request Body Objects .
headers	Map[string, Header Object Reference Object]	An object to hold reusable Header Objects .
securitySchemes	Map[string, Security Scheme Object Reference Object]	An object to hold reusable Security Scheme Objects .
links	Map[string, Link Object Reference Object]	An object to hold reusable Link Objects .
callbacks	Map[string, Callback Object Reference Object]	An object to hold reusable Callback Objects .

This object MAY be extended with [Specification Extensions](#).

All the fixed fields declared above are objects that MUST use keys that match the regular expression: `^[a-zA-Z0-9\.\-_]+$`.

Field Name Examples:

```
User
User_1
User_Name
```

user-name
my.org.User

Components Object Example

```
"components": {
  "schemas": {
    "GeneralError": {
      "type": "object",
      "properties": {
        "code": {
          "type": "integer",
          "format": "int32"
        },
        "message": {
          "type": "string"
        }
      }
    },
    "Category": {
      "type": "object",
      "properties": {
        "id": {
          "type": "integer",
          "format": "int64"
        },
        "name": {
          "type": "string"
        }
      }
    },
    "Tag": {
      "type": "object",
      "properties": {
        "id": {
          "type": "integer",
          "format": "int64"
        },
        "name": {
          "type": "string"
        }
      }
    }
  },
  "parameters": {
    "skipParam": {
      "name": "skip",
      "in": "query",
      "description": "number of items to skip",
      "required": true,
      "schema": {
        "type": "integer",
        "format": "int32"
      }
    },
    "limitParam": {
      "name": "limit",
      "in": "query",
```

```

        "description": "max records to return",
        "required": true,
        "schema": {
            "type": "integer",
            "format": "int32"
        }
    },
    "responses": {
        "NotFound": {
            "description": "Entity not found."
        },
        "IllegalInput": {
            "description": "Illegal input for operation."
        },
        "GeneralError": {
            "description": "General Error",
            "content": {
                "application/json": {
                    "schema": {
                        "$ref": "#/components/schemas/GeneralError"
                    }
                }
            }
        }
    },
    "securitySchemes": {
        "api_key": {
            "type": "apiKey",
            "name": "api_key",
            "in": "header"
        },
        "petstore_auth": {
            "type": "oauth2",
            "flows": {
                "implicit": {
                    "authorizationUrl": "http://example.org/api/oauth/dialog",
                    "scopes": {
                        "write:pets": "modify pets in your account",
                        "read:pets": "read your pets"
                    }
                }
            }
        }
    }
},
components:
  schemas:
    GeneralError:
      type: object
      properties:
        code:
          type: integer
          format: int32
        message:
          type: string
    Category:
      type: object
      properties:

```

```

    id:
      type: integer
      format: int64
    name:
      type: string
  Tag:
    type: object
    properties:
      id:
        type: integer
        format: int64
      name:
        type: string
  parameters:
    skipParam:
      name: skip
      in: query
      description: number of items to skip
      required: true
      schema:
        type: integer
        format: int32
    limitParam:
      name: limit
      in: query
      description: max records to return
      required: true
      schema:
        type: integer
        format: int32
  responses:
    NotFound:
      description: Entity not found.
    IllegalInput:
      description: Illegal input for operation.
    GeneralError:
      description: General Error
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/GeneralError'
  securitySchemes:
    api_key:
      type: apiKey
      name: api_key
      in: header
    petstore_auth:
      type: oauth2
      flows:
        implicit:
          authorizationUrl: http://example.org/api/oauth/dialog
          scopes:
            write:pets: modify pets in your account
            read:pets: read your pets

```

Paths Object

Holds the relative paths to the individual endpoints and their operations. The path is appended to the URL from the [Server Object](#) 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 (no relative URL resolution) to the expanded URL from the Server Object's url field in order to construct the full URL. Path templating is allowed. When matching URLs, concrete (non-templated) paths would be matched before their templated counterparts. Templated paths with the same hierarchy but different templated names MUST NOT exist as they are identical. In case of ambiguous matching, it's up to the tooling to decide which one to use.

This object MAY be extended with [Specification Extensions](#).

Path Templating Matching

Assuming the following paths, the concrete definition, /pets/mine, will be matched first if used:

```
/pets/{petId}
/pets/mine
```

The following paths are considered identical and invalid:

```
/pets/{petId}
/pets/{name}
```

The following may lead to ambiguous resolution:

```
/ {entity} /me
/books/{id}
```

Paths Object Example

```
{
  "/pets": {
    "get": {
      "description": "Returns all pets from the system that the user has access to",
      "responses": {
        "200": {
          "description": "A list of pets.",
          "content": {
            "application/json": {
              "schema": {
                "type": "array",
                "items": {
```


Field Name	Type	Description
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.
trace	Operation Object	A definition of a TRACE operation on this path.
servers	[Server Object]	An alternative server array to service all operations in 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 OpenAPI Object's components/parameters .

This object MAY be extended with [Specification Extensions](#).

Path Item Object Example

```
{
  "get": {
    "description": "Returns pets based on ID",
    "summary": "Find pets by ID",
    "operationId": "getPetsById",
    "responses": {
      "200": {
        "description": "pet response",
        "content": {
          "*/*": {
            "schema": {
              "type": "array",
              "items": {
```

```

        "$ref": "#/components/schemas/Pet"
      }
    }
  },
  "default": {
    "description": "error payload",
    "content": {
      "text/html": {
        "schema": {
          "$ref": "#/components/schemas/ErrorMessage"
        }
      }
    }
  }
},
"parameters": [
  {
    "name": "id",
    "in": "path",
    "description": "ID of pet to use",
    "required": true,
    "schema": {
      "type": "array",
      "items": {
        "type": "string"
      }
    }
  },
  {
    "style": "simple"
  }
]
}
get:
  description: Returns pets based on ID
  summary: Find pets by ID
  operationId: getPetsById
  responses:
    '200':
      description: pet response
      content:
        '*/*':
          schema:
            type: array
            items:
              $ref: '#/components/schemas/Pet'
      default:
        description: error payload
        content:
          'text/html':
            schema:
              $ref: '#/components/schemas/ErrorMessage'
  parameters:
    - name: id
      in: path
      description: ID of pet to use
      required: true
      schema:

```

```
type: array
style: simple
items:
  type: string
```

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.
description	string	A verbose explanation of the operation behavior. CommonMark syntax MAY be used for rich text representation.
externalDocs	External Documentation Object	Additional external documentation for this operation.
operationId	string	Unique string used to identify the operation. The id MUST be unique among all operations described in the API. The operationId value is case-sensitive . Tools and libraries MAY use the operationId to uniquely identify an operation, therefore, it is RECOMMENDED to follow common programming naming conventions.
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 OpenAPI Object's components/parameters .

Field Name	Type	Description
requestBody	Request Body Object Reference Object	The request body applicable for this operation. The requestBody is only supported in HTTP methods where the HTTP 1.1 specification RFC7231 has explicitly defined semantics for request bodies. In other cases where the HTTP spec is vague, requestBody SHALL be ignored by consumers.
responses	Responses Object	REQUIRED. The list of possible responses as they are returned from executing this operation.
callbacks	Map[string, Callback Object Reference Object]	A map of possible out-of band callbacks related to the parent operation. The key is a unique identifier for the Callback Object. Each value in the map is a Callback Object that describes a request that may be initiated by the API provider and the expected responses. The key value used to identify the callback object is an expression, evaluated at runtime, that identifies a URL to use for the callback operation.
deprecated	boolean	Declares this operation to be deprecated. Consumers SHOULD refrain from usage of the declared operation. Default value is false.
security	[Security Requirement Object]	A declaration of which security mechanisms can be used for this operation. The list of values includes alternative security requirement objects that can be used. Only one of the security requirement objects need to be satisfied to authorize a request. This definition overrides any declared top-level security . To remove a top-level security declaration, an empty array can be used.
servers	[Server Object]	An alternative server array to service this operation. If an alternative server object is specified at the Path Item Object or Root level, it will be overridden by this value.

This object MAY be extended with [Specification Extensions](#).

Operation Object Example

```

{
  "tags": [
    "pet"
  ],
  "summary": "Updates a pet in the store with form data",
  "operationId": "updatePetWithForm",
  "parameters": [
    {
      "name": "petId",
      "in": "path",
      "description": "ID of pet that needs to be updated",
      "required": true,
      "schema": {
        "type": "string"
      }
    }
  ],
  "requestBody": {
    "content": {
      "application/x-www-form-urlencoded": {
        "schema": {
          "type": "object",
          "properties": {
            "name": {
              "description": "Updated name of the pet",
              "type": "string"
            },
            "status": {
              "description": "Updated status of the pet",
              "type": "string"
            }
          }
        },
        "required": ["status"]
      }
    }
  },
  "responses": {
    "200": {
      "description": "Pet updated.",
      "content": {
        "application/json": {},
        "application/xml": {}
      }
    },
    "405": {
      "description": "Method Not Allowed",
      "content": {
        "application/json": {},
        "application/xml": {}
      }
    }
  },
  "security": [
    {
      "petstore_auth": [
        "write:pets",
        "read:pets"
      ]
    }
  ]
}

```

```

    }
  ]
}
tags:
- pet
summary: Updates a pet in the store with form data
operationId: updatePetWithForm
parameters:
- name: petId
  in: path
  description: ID of pet that needs to be updated
  required: true
  schema:
    type: string
requestBody:
  content:
    'application/x-www-form-urlencoded':
      schema:
        properties:
          name:
            description: Updated name of the pet
            type: string
          status:
            description: Updated status of the pet
            type: string
        required:
          - status
responses:
  '200':
    description: Pet updated.
    content:
      'application/json': {}
      'application/xml': {}
  '405':
    description: Method Not Allowed
    content:
      'application/json': {}
      'application/xml': {}
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. CommonMark syntax MAY be used for rich text representation.
url	string	REQUIRED. The URL for the target documentation. Value MUST be in the format of a URL.

This object MAY be extended with [Specification Extensions](#).

External Documentation Object Example

```
{
  "description": "Find more info here",
  "url": "https://example.com"
}
description: Find more info here
url: https://example.com
```

Parameter Object

Describes a single operation parameter.

A unique parameter is defined by a combination of a [name](#) and [location](#).

Parameter Locations

There are four possible parameter locations specified by the `in` field:

- `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. Note that [RFC7230](#) states header names are case insensitive.
- `cookie` - Used to pass a specific cookie value to the API.

Fixed Fields

Field Name	Type	Description
name	string	<p>REQUIRED. The name of the parameter. Parameter names are <i>case sensitive</i>.</p> <ul style="list-style-type: none"> 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. If in is "header" and the name field is "Accept", "Content-Type" or "Authorization", the parameter definition SHALL be ignored. For all other cases, the name corresponds to the parameter name used by the in property.
in	string	<p>REQUIRED. The location of the parameter. Possible values are "query", "header", "path" or "cookie".</p>
description	string	<p>A brief description of the parameter. This could contain examples of use. CommonMark syntax MAY be used for rich text representation.</p>
required	boolean	<p>Determines whether this parameter is mandatory. If the parameter location is "path", this property is REQUIRED and its value MUST be true. Otherwise, the property MAY be included and its default value is false.</p>
deprecated	boolean	<p>Specifies that a parameter is deprecated and SHOULD be transitioned out of usage. Default value is false.</p>
allowEmptyValue	boolean	<p>Sets the ability to pass empty-valued parameters. This is valid only for query parameters and allows sending a parameter with an empty value. Default value is false. If style is used, and if behavior is n/a (cannot be serialized), the value of allowEmptyValue SHALL be ignored. Use of this property is NOT RECOMMENDED, as it is likely to be removed in a later revision.</p>

The rules for serialization of the parameter are specified in one of two ways. For simpler scenarios, a [schema](#) and [style](#) can describe the structure and syntax of the parameter.

Field Name	Type	Description
style	string	Describes how the parameter value will be serialized depending on the type of the parameter value. Default values (based on value of in): for query - form; for path - simple; for header - simple; for cookie - form.
explode	boolean	When this is true, parameter values of type array or object generate separate parameters for each value of the array or key-value pair of the map. For other types of parameters this property has no effect. When style is form, the default value is true. For all other styles, the default value is false.
allowReserved	boolean	Determines whether the parameter value SHOULD allow reserved characters, as defined by RFC3986 :/?#[]@!\$&'()*+,-;= to be included without percent-encoding. This property only applies to parameters with an in value of query. The default value is false.
schema	Schema Object Reference Object	The schema defining the type used for the parameter.
example	Any	Example of the media type. The example SHOULD match the specified schema and encoding properties if present. The example field is mutually exclusive of the examples field. Furthermore, if referencing a schema which contains an example, the example value SHALL <i>override</i> the example provided by the schema. To represent examples of media types that cannot naturally be represented in JSON or YAML, a string value can contain the example with escaping where necessary.
examples	Map[string, Example Object Reference Object]	Examples of the media type. Each example SHOULD contain a value in the correct format as specified in the parameter encoding. The examples field is mutually exclusive of the example field. Furthermore, if referencing a schema which contains an example,

Field Name	Type	Description
		the examples value SHALL <i>override</i> the example provided by the schema.

For more complex scenarios, the [content](#) property can define the media type and schema of the parameter. A parameter MUST contain either a schema property, or a content property, but not both. When `example` or `examples` are provided in conjunction with the schema object, the example MUST follow the prescribed serialization strategy for the parameter.

Field Name	Type	Description
content	Map[string, Media Type Object]	A map containing the representations for the parameter. The key is the media type and the value describes it. The map MUST only contain one entry.

Style Values

In order to support common ways of serializing simple parameters, a set of style values are defined.

style	type	in	Comments
matrix	primitive, array, object	path	Path-style parameters defined by RFC6570
label	primitive, array, object	path	Label style parameters defined by RFC6570
form	primitive, array, object	query, cookie	Form style parameters defined by RFC6570 . This option replaces <code>collectionFormat</code> with a <code>csv</code> (when <code>explode</code> is false) or <code>multi</code> (when <code>explode</code> is true) value from OpenAPI 2.0.

style	type	in	Comments
simple	array	path, header	Simple style parameters defined by RFC6570 . This option replaces collectionFormat with a csv value from OpenAPI 2.0.
spaceDelimited	array	query	Space separated array values. This option replaces collectionFormat equal to ssv from OpenAPI 2.0.
pipeDelimited	array	query	Pipe separated array values. This option replaces collectionFormat equal to pipes from OpenAPI 2.0.
deepObject	object	query	Provides a simple way of rendering nested objects using form parameters.

Style Examples

Assume a parameter named `color` has one of the following values:

```
string -> "blue"
array -> ["blue","black","brown"]
object -> { "R": 100, "G": 200, "B": 150 }
```

The following table shows examples of rendering differences for each value.

style	explode	empty	string	array	object
matrix	false	;color	;color=blue	;color=blue,black,brown	;color=R,100,G,200,B,150
matrix	true	;color	;color=blue	;color=blue;color=black;color=brown	;R=100;G=200;B=150
label	false	.	.blue	.blue.black.brown	.R.100.G.200.B.150

style	explode	empty	string	array	object
label	true	.	.blue	.blue.black.brown	.R=100.G=200.B=150
form	false	color=	color=blue	color=blue,black,brown	color=R,100,G,200,B,150
form	true	color=	color=blue	color=blue&color=black&color=brown	R=100&G=200&B=150
simple	false	n/a	blue	blue,black,brown	R,100,G,200,B,150
simple	true	n/a	blue	blue,black,brown	R=100,G=200,B=150
spaceDelimited	false	n/a	n/a	blue%20black%20brown	R%20100%20G%20200%20B%20150
pipeDelimited	false	n/a	n/a	blue black brown	R 100 G 200
deepObject	true	n/a	n/a	n/a	color[R]=100&color[G]=200&color[B]=150

This object MAY be extended with [Specification Extensions](#).

Parameter Object Examples

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,
  "schema": {
    "type": "array",
    "items": {
      "type": "integer",
      "format": "int64"
    }
  },
  "style": "simple"
}
```

name: token
in: header

```
description: token to be passed as a header
required: true
schema:
  type: array
  items:
    type: integer
    format: int64
style: simple
```

A path parameter of a string value:

```
{
  "name": "username",
  "in": "path",
  "description": "username to fetch",
  "required": true,
  "schema": {
    "type": "string"
  }
}
name: username
in: path
description: username to fetch
required: true
schema:
  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,
  "schema": {
    "type": "array",
    "items": {
      "type": "string"
    }
  },
  "style": "form",
  "explode": true
}
name: id
in: query
description: ID of the object to fetch
required: false
schema:
  type: array
  items:
    type: string
style: form
explode: true
```

A free-form query parameter, allowing undefined parameters of a specific type:

```
{
  "in": "query",
  "name": "freeForm",
  "schema": {
```

```

    "type": "object",
    "additionalProperties": {
      "type": "integer"
    },
  },
  "style": "form"
}
in: query
name: freeForm
schema:
  type: object
  additionalProperties:
    type: integer
style: form

```

A complex parameter using content to define serialization:

```

{
  "in": "query",
  "name": "coordinates",
  "content": {
    "application/json": {
      "schema": {
        "type": "object",
        "required": [
          "lat",
          "long"
        ],
        "properties": {
          "lat": {
            "type": "number"
          },
          "long": {
            "type": "number"
          }
        }
      }
    }
  }
}
in: query
name: coordinates
content:
  application/json:
    schema:
      type: object
      required:
        - lat
        - long
      properties:
        lat:
          type: number
        long:
          type: number

```

Request Body Object

Describes a single request body.

Fixed Fields

Field Name	Type	Description
description	string	A brief description of the request body. This could contain examples of use. CommonMark syntax MAY be used for rich text representation.
content	Map[string, Media Type Object]	REQUIRED. The content of the request body. The key is a media type or media type range and the value describes it. For requests that match multiple keys, only the most specific key is applicable. e.g. text/plain overrides text/*
required	boolean	Determines if the request body is required in the request. Defaults to false.

This object MAY be extended with [Specification Extensions](#).

Request Body Examples

A request body with a referenced model definition.

```
{
  "description": "user to add to the system",
  "content": {
    "application/json": {
      "schema": {
        "$ref": "#/components/schemas/User"
      },
      "examples": {
        "user": {
          "summary": "User Example",
          "externalValue": "http://foo.bar/examples/user-example.json"
        }
      }
    },
    "application/xml": {
      "schema": {
        "$ref": "#/components/schemas/User"
      },
      "examples": {
        "user": {
          "summary": "User example in XML",
          "externalValue": "http://foo.bar/examples/user-example.xml"
        }
      }
    },
    "text/plain": {
      "examples": {
        "user": {
```

```

        "summary": "User example in Plain text",
        "externalValue": "http://foo.bar/examples/user-example.txt"
    }
}
},
"/": {
    "examples": {
        "user": {
            "summary": "User example in other format",
            "externalValue": "http://foo.bar/examples/user-example.whatever"
        }
    }
}
}
}
}
description: user to add to the system
content:
  'application/json':
    schema:
      $ref: '#/components/schemas/User'
    examples:
      user:
        summary: User Example
        externalValue: 'http://foo.bar/examples/user-example.json'
  'application/xml':
    schema:
      $ref: '#/components/schemas/User'
    examples:
      user:
        summary: User Example in XML
        externalValue: 'http://foo.bar/examples/user-example.xml'
  'text/plain':
    examples:
      user:
        summary: User example in text plain format
        externalValue: 'http://foo.bar/examples/user-example.txt'
  '*/*':
    examples:
      user:
        summary: User example in other format
        externalValue: 'http://foo.bar/examples/user-example.whatever'

```

A body parameter that is an array of string values:

```

{
  "description": "user to add to the system",
  "content": {
    "text/plain": {
      "schema": {
        "type": "array",
        "items": {
          "type": "string"
        }
      }
    }
  }
}
}
description: user to add to the system
required: true
content:

```

```
text/plain:
  schema:
    type: array
    items:
      type: string
```

Media Type Object

Each Media Type Object provides schema and examples for the media type identified by its key.

Fixed Fields

Field Name	Type	Description
schema	Schema Object Reference Object	The schema defining the content of the request, response, or parameter.
example	Any	Example of the media type. The example object SHOULD be in the correct format as specified by the media type. The example field is mutually exclusive of the examples field. Furthermore, if referencing a schema which contains an example, the example value SHALL <i>override</i> the example provided by the schema.
examples	Map[string, Example Object Reference Object]	Examples of the media type. Each example object SHOULD match the media type and specified schema if present. The examples field is mutually exclusive of the example field. Furthermore, if referencing a schema which contains an example, the examples value SHALL <i>override</i> the example provided by the schema.
encoding	Map[string, Encoding Object]	A map between a property name and its encoding information. The key, being the property name, MUST exist in the schema as a property. The encoding object SHALL only apply to requestBody objects when the media type is multipart or application/x-www-form-urlencoded.

This object MAY be extended with [Specification Extensions](#).

Media Type Examples

```
{
```

```

"application/json": {
  "schema": {
    "$ref": "#/components/schemas/Pet"
  },
  "examples": {
    "cat": {
      "summary": "An example of a cat",
      "value": {
        "name": "Fluffy",
        "petType": "Cat",
        "color": "White",
        "gender": "male",
        "breed": "Persian"
      }
    },
    "dog": {
      "summary": "An example of a dog with a cat's name",
      "value": {
        "name": "Puma",
        "petType": "Dog",
        "color": "Black",
        "gender": "Female",
        "breed": "Mixed"
      }
    },
    "frog": {
      "$ref": "#/components/examples/frog-example"
    }
  }
}
}
application/json:
schema:
  $ref: "#/components/schemas/Pet"
examples:
  cat:
    summary: An example of a cat
    value:
      name: Fluffy
      petType: Cat
      color: White
      gender: male
      breed: Persian
  dog:
    summary: An example of a dog with a cat's name
    value:
      name: Puma
      petType: Dog
      color: Black
      gender: Female
      breed: Mixed
  frog:
    $ref: "#/components/examples/frog-example"

```

Considerations for File Uploads

In contrast with the 2.0 specification, file input/output content in OpenAPI is described with the same semantics as any other schema type. Specifically:

```
# content transferred with base64 encoding
schema:
  type: string
  format: base64
# content transferred in binary (octet-stream):
schema:
  type: string
  format: binary
```

These examples apply to either input payloads of file uploads or response payloads.

A requestBody for submitting a file in a POST operation may look like the following example:

```
requestBody:
  content:
    application/octet-stream:
      # any media type is accepted, functionally equivalent to `*/*`
      schema:
        # a binary file of any type
        type: string
        format: binary
```

In addition, specific media types MAY be specified:

```
# multiple, specific media types may be specified:
requestBody:
  content:
    # a binary file of type png or jpeg
    'image/jpeg':
      schema:
        type: string
        format: binary
    'image/png':
      schema:
        type: string
        format: binary
```

To upload multiple files, a multipart media type MUST be used:

```
requestBody:
  content:
    multipart/form-data:
      schema:
        properties:
          # The property name 'file' will be used for all files.
          file:
            type: array
            items:
              type: string
              format: binary
```

Support for x-www-form-urlencoded Request Bodies

To submit content using form url encoding via [RFC1866](#), the following definition may be used:

```
requestBody:
```

```

content:
  application/x-www-form-urlencoded:
    schema:
      type: object
      properties:
        id:
          type: string
          format: uuid
        address:
          # complex types are stringified to support RFC 1866
          type: object
          properties: {}

```

In this example, the contents in the `requestBody` MUST be stringified per [RFC1866](#) when passed to the server. In addition, the `address` field complex object will be stringified.

When passing complex objects in the `application/x-www-form-urlencoded` content type, the default serialization strategy of such properties is described in the [Encoding Object's](#) `style` property as `form`.

Special Considerations for multipart Content

It is common to use `multipart/form-data` as a Content-Type when transferring request bodies to operations. In contrast to 2.0, a schema is REQUIRED to define the input parameters to the operation when using multipart content. This supports complex structures as well as supporting mechanisms for multiple file uploads.

When passing in multipart types, boundaries MAY be used to separate sections of the content being transferred — thus, the following default Content-Types are defined for multipart:

- If the property is a primitive, or an array of primitive values, the default Content-Type is `text/plain`
- If the property is complex, or an array of complex values, the default Content-Type is `application/json`
- If the property is a `type: string` with `format: binary` or `format: base64` (aka a file object), the default Content-Type is `application/octet-stream`

Examples:

```

requestBody:
  content:
    multipart/form-data:
      schema:
        type: object
        properties:
          id:
            type: string
            format: uuid
          address:
            # default Content-Type for objects is `application/json`
            type: object
            properties: {}
          profileImage:
            # default Content-Type for string/binary is `application/octet-stream`
            type: string

```

```

    format: binary
  children:
    # default Content-Type for arrays is based on the `inner` type
    (text/plain here)
    type: array
    items:
      type: string
  addresses:
    # default Content-Type for arrays is based on the `inner` type (object
    shown, so `application/json` in this example)
    type: array
    items:
      type: '#/components/schemas/Address'

```

An encoding attribute is introduced to give you control over the serialization of parts of multipart request bodies. This attribute is *only* applicable to multipart and application/x-www-form-urlencoded request bodies.

Encoding Object

A single encoding definition applied to a single schema property.

Fixed Fields

Field Name	Type	Description
contentType	string	The Content-Type for encoding a specific property. Default value depends on the property type: for string with format being binary – application/octet-stream; for other primitive types – text/plain; for object - application/json; for array – the default is defined based on the inner type. The value can be a specific media type (e.g. application/json), a wildcard media type (e.g. image/*), or a comma-separated list of the two types.
headers	Map[string, Header Object Reference Object]	A map allowing additional information to be provided as headers, for example Content-Disposition. Content-Type is described separately and SHALL be ignored in this section. This property SHALL be ignored if the request body media type is not a multipart.
style	string	Describes how a specific property value will be serialized depending on its type. See Parameter Object for details on the <code>style</code> property. The behavior follows the same values as query parameters, including default values. This

Field Name	Type	Description
		property SHALL be ignored if the request body media type is not application/x-www-form-urlencoded.
explode	boolean	When this is true, property values of type array or object generate separate parameters for each value of the array, or key-value-pair of the map. For other types of properties this property has no effect. When style is form, the default value is true. For all other styles, the default value is false. This property SHALL be ignored if the request body media type is not application/x-www-form-urlencoded.
allowReserved	boolean	Determines whether the parameter value SHOULD allow reserved characters, as defined by RFC3986 : / ? # [] @ ! \$ & ' () * + , ; = to be included without percent-encoding. The default value is false. This property SHALL be ignored if the request body media type is not application/x-www-form-urlencoded.

This object MAY be extended with [Specification Extensions](#).

Encoding Object Example

```
requestBody:
  content:
    multipart/mixed:
      schema:
        type: object
        properties:
          id:
            # default is text/plain
            type: string
            format: uuid
          address:
            # default is application/json
            type: object
            properties: {}
          historyMetadata:
            # need to declare XML format!
            description: metadata in XML format
            type: object
            properties: {}
          profileImage:
            # default is application/octet-stream, need to declare an image type
            only!
```



```
    type: string
    format: binary
  encoding:
    historyMetadata:
      # require XML Content-Type in utf-8 encoding
      contentType: application/xml; charset=utf-8
    profileImage:
      # only accept png/jpeg
      contentType: image/png, image/jpeg
    headers:
      X-Rate-Limit-Limit:
        description: The number of allowed requests in the current period
        schema:
          type: integer
```

Responses Object

A container for the expected responses of an operation. The container maps a HTTP response code to the expected response.

The documentation is not necessarily expected to cover all possible HTTP response codes because they may not be known in advance. However, documentation is expected to cover a successful operation response and any known errors.

The default MAY be used as 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. Use this field to cover undeclared responses. A Reference Object can link to a response that the OpenAPI Object's components/responses section defines.

Patterned Fields

Field Pattern	Type	Description
HTTP Status Code	Response Object Reference Object	Any HTTP status code can be used as the property name, but only one property per code, to describe the expected response for that HTTP status code. A Reference Object can link to a response that is defined in the OpenAPI Object's components/responses section. This field MUST be enclosed in quotation marks (for example, "200") for compatibility between JSON and YAML. To define a range of response codes, this field MAY contain the uppercase wildcard character X. For example, 2XX represents all response codes between [200-299]. Only the following range definitions are allowed: 1XX, 2XX, 3XX, 4XX, and 5XX. If a response is defined using an explicit code, the explicit code definition takes precedence over the range definition for that code.

This object MAY be extended with [Specification Extensions](#).

Responses Object Example

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

```
{
  "200": {
    "description": "a pet to be returned",
    "content": {
      "application/json": {
        "schema": {
          "$ref": "#/components/schemas/Pet"
        }
      }
    }
  },
  "default": {
    "description": "Unexpected error",
    "content": {
      "application/json": {
        "schema": {
          "$ref": "#/components/schemas/ErrorMessage"
        }
      }
    }
  }
}
'200':
description: a pet to be returned
content:
  application/json:
    schema:
```

```

    $ref: '#/components/schemas/Pet'
default:
  description: Unexpected error
  content:
    application/json:
      schema:
        $ref: '#/components/schemas/ErrorResponse'

```

Response Object

Describes a single response from an API Operation, including design-time, static links to operations based on the response.

Fixed Fields

Field Name	Type	Description
description	string	REQUIRED. A short description of the response. CommonMark syntax MAY be used for rich text representation.
headers	Map[string, Header Object Reference Object]	Maps a header name to its definition. RFC7230 states header names are case insensitive. If a response header is defined with the name "Content-Type", it SHALL be ignored.
content	Map[string, Media Type Object]	A map containing descriptions of potential response payloads. The key is a media type or media type range and the value describes it. For responses that match multiple keys, only the most specific key is applicable. e.g. text/plain overrides text/*
links	Map[string, Link Object Reference Object]	A map of operations links that can be followed from the response. The key of the map is a short name for the link, following the naming constraints of the names for Component Objects .

This object MAY be extended with [Specification Extensions](#).

Response Object Examples

Response of an array of a complex type:

```

{
  "description": "A complex object array response",

```

```

"content": {
  "application/json": {
    "schema": {
      "type": "array",
      "items": {
        "$ref": "#/components/schemas/VeryComplexType"
      }
    }
  }
}
}
description: A complex object array response
content:
  application/json:
    schema:
      type: array
      items:
        $ref: '#/components/schemas/VeryComplexType'

```

Response with a string type:

```

{
  "description": "A simple string response",
  "content": {
    "text/plain": {
      "schema": {
        "type": "string"
      }
    }
  }
}
description: A simple string response
content:
  text/plain:
    schema:
      type: string

```

Plain text response with headers:

```

{
  "description": "A simple string response",
  "content": {
    "text/plain": {
      "schema": {
        "type": "string"
      }
    }
  },
  "headers": {
    "X-Rate-Limit-Limit": {
      "description": "The number of allowed requests in the current period",
      "schema": {
        "type": "integer"
      }
    },
    "X-Rate-Limit-Remaining": {
      "description": "The number of remaining requests in the current period",
      "schema": {
        "type": "integer"
      }
    }
  }
}

```

```

    }
  },
  "X-Rate-Limit-Reset": {
    "description": "The number of seconds left in the current period",
    "schema": {
      "type": "integer"
    }
  }
}
}
description: A simple string response
content:
  text/plain:
    schema:
      type: string
      example: 'whoa!'
headers:
  X-Rate-Limit-Limit:
    description: The number of allowed requests in the current period
    schema:
      type: integer
  X-Rate-Limit-Remaining:
    description: The number of remaining requests in the current period
    schema:
      type: integer
  X-Rate-Limit-Reset:
    description: The number of seconds left in the current period
    schema:
      type: integer

```

Response with no return value:

```

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

```

Callback Object

A map of possible out-of band callbacks related to the parent operation. Each value in the map is a [Path Item Object](#) that describes a set of requests that may be initiated by the API provider and the expected responses. The key value used to identify the callback object is an expression, evaluated at runtime, that identifies a URL to use for the callback operation.

Patterned Fields

Field Pattern	Type	Description
{expression}	Path Item Object	A Path Item Object used to define a callback request and expected responses. A complete example is available.

This object MAY be extended with [Specification Extensions](#).

[Key Expression](#)

The key that identifies the [Path Item Object](#) is a [runtime expression](#) that can be evaluated in the context of a runtime HTTP request/response to identify the URL to be used for the callback request. A simple example might be `$request.body#/url`. However, using a [runtime expression](#) the complete HTTP message can be accessed. This includes accessing any part of a body that a JSON Pointer [RFC6901](#) can reference.

For example, given the following HTTP request:

```
POST /subscribe/myevent?queryUrl=http://clientdomain.com/stillrunning HTTP/1.1
Host: example.org
Content-Type: application/json
Content-Length: 187
```

```
{
  "failedUrl" : "http://clientdomain.com/failed",
  "successUrls" : [
    "http://clientdomain.com/fast",
    "http://clientdomain.com/medium",
    "http://clientdomain.com/slow"
  ]
}
```

201 Created

Location: <http://example.org/subscription/1>

The following examples show how the various expressions evaluate, assuming the callback operation has a path parameter named `eventType` and a query parameter named `queryUrl`.

Expression	Value
<code>\$url</code>	http://example.org/subscribe/myevent?queryUrl=http://clientdomain.com/stillrunning
<code>\$method</code>	POST
<code>\$request.path.eventType</code>	myevent
<code>\$request.query.queryUrl</code>	http://clientdomain.com/stillrunning

Expression	Value
\$request.header.content-Type	application/json
\$request.body#/failedUrl	http://clientdomain.com/failed
\$request.body#/successUrls/2	http://clientdomain.com/medium
\$response.header.Location	http://example.org/subscription/1

Callback Object Example

The following example shows a callback to the URL specified by the `id` and `email` property in the request body.

myWebhook:

```
'http://notificationServer.com?transactionId={$request.body#/id}&email={$request.body#/email}':
  post:
    requestBody:
      description: Callback payload
      content:
        'application/json':
          schema:
            $ref: '#/components/schemas/SomePayload'
    responses:
      '200':
        description: webhook successfully processed and no retries will be performed
```

Example Object

Fixed Fields

Field Name	Type	Description
summary	string	Short description for the example.

Field Name	Type	Description
description	string	Long description for the example. CommonMark syntax MAY be used for rich text representation.
value	Any	Embedded literal example. The value field and externalValue field are mutually exclusive. To represent examples of media types that cannot naturally be represented in JSON or YAML, use a string value to contain the example, escaping where necessary.
externalValue	string	A URL that points to the literal example. This provides the capability to reference examples that cannot easily be included in JSON or YAML documents. The value field and externalValue field are mutually exclusive.

This object MAY be extended with [Specification Extensions](#).

In all cases, the example value is expected to be compatible with the type schema of its associated value. Tooling implementations MAY choose to validate compatibility automatically, and reject the example value(s) if incompatible.

Example Object Examples

In a model:

```
schemas:
  properties:
    name:
      type: string
      examples:
        name:
          $ref: http://example.org/petapi-
examples/openapi.json#/components/examples/name-example
```

In a request body:

```
requestBody:
  content:
    'application/json':
      schema:
        $ref: '#/components/schemas/Address'
      examples:
        foo:
          summary: A foo example
          value: {"foo": "bar"}
        bar:
          summary: A bar example
          value: {"bar": "baz"}
    'application/xml':
      examples:
        xmlExample:
```



```

summary: This is an example in XML
externalValue: 'http://example.org/examples/address-example.xml'
'text/plain':
  examples:
    textExample:
      summary: This is a text example
      externalValue: 'http://foo.bar/examples/address-example.txt'

```

In a parameter:

```

parameters:
  - name: 'zipCode'
    in: 'query'
    schema:
      type: 'string'
      format: 'zip-code'
      examples:
        zip-example:
          $ref: '#/components/examples/zip-example'

```

In a response:

```

responses:
  '200':
    description: your car appointment has been booked
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/SuccessResponse'
        examples:
          confirmation-success:
            $ref: '#/components/examples/confirmation-success'

```

Link Object

The `Link` object represents a possible design-time link for a response. The presence of a link does not guarantee the caller's ability to successfully invoke it, rather it provides a known relationship and traversal mechanism between responses and other operations.

Unlike *dynamic* links (i.e. links provided **in** the response payload), the OAS linking mechanism does not require link information in the runtime response.

For computing links, and providing instructions to execute them, a [runtime expression](#) is used for accessing values in an operation and using them as parameters while invoking the linked operation.

Fixed Fields

Field Name	Type	Description
operationRef	string	A relative or absolute reference to an OAS operation. This field is mutually exclusive of the operationId field, and MUST point to an Operation Object . Relative operationRef values MAY be used to locate an existing Operation Object in the OpenAPI definition.
operationId	string	The name of an <i>existing</i> , resolvable OAS operation, as defined with a unique operationId. This field is mutually exclusive of the operationRef field.
parameters	Map[string, Any {expression}]	A map representing parameters to pass to an operation as specified with operationId or identified via operationRef. The key is the parameter name to be used, whereas the value can be a constant or an expression to be evaluated and passed to the linked operation. The parameter name can be qualified using the parameter location [{in}].[name] for operations that use the same parameter name in different locations (e.g. path.id).
requestBody	Any {expression}	A literal value or {expression} to use as a request body when calling the target operation.
description	string	A description of the link. CommonMark syntax MAY be used for rich text representation.
server	Server Object	A server object to be used by the target operation.

This object MAY be extended with [Specification Extensions](#).

A linked operation MUST be identified using either an operationRef or operationId. In the case of an operationId, it MUST be unique and resolved in the scope of the OAS document. Because of the potential for name clashes, the operationRef syntax is preferred for specifications with external references.

Examples

Computing a link from a request operation where the \$request.path.id is used to pass a request parameter to the linked operation.

```
paths:
  /users/{id}:
    parameters:
      - name: id
        in: path
        required: true
        description: the user identifier, as userId
```

```

    schema:
      type: string
  get:
    responses:
      '200':
        description: the user being returned
        content:
          application/json:
            schema:
              type: object
              properties:
                uuid: # the unique user id
                  type: string
                  format: uuid
        links:
          address:
            # the target link operationId
            operationId: getUserAddress
            parameters:
              # get the `id` field from the request path parameter named `id`
              userId: $request.path.id
# the path item of the linked operation
/users/{userid}/address:
  parameters:
    - name: userid
      in: path
      required: true
      description: the user identifier, as userId
      schema:
        type: string
# linked operation
  get:
    operationId: getUserAddress
    responses:
      '200':
        description: the user's address

```

When a runtime expression fails to evaluate, no parameter value is passed to the target operation.

Values from the response body can be used to drive a linked operation.

```

links:
  address:
    operationId: getUserAddressByUUID
    parameters:
      # get the `uuid` field from the `uuid` field in the response body
      userUuid: $response.body#/uuid

```

Clients follow all links at their discretion. Neither permissions, nor the capability to make a successful call to that link, is guaranteed solely by the existence of a relationship.

OperationRef Examples

As references to operationId MAY NOT be possible (the operationId is an optional value), references MAY also be made through a relative operationRef:

```

links:
  UserRepositories:
    # returns array of '#/components/schemas/repository'
    operationRef: '#/paths/~12.0~1repositories~1{username}/get'
    parameters:
      username: $response.body#/username

```

or an absolute operationRef:

```

links:
  UserRepositories:
    # returns array of '#/components/schemas/repository'
    operationRef: 'https://na2.gigantic-
server.com/#!/paths/~12.0~1repositories~1{username}/get'
    parameters:
      username: $response.body#/username

```

Note that in the use of operationRef, the *escaped forward-slash* is necessary when using JSON references.

Runtime Expressions

Runtime expressions allow defining values based on information that will only be available within the HTTP message in an actual API call. This mechanism is used by [Link Objects](#) and [Callback Objects](#).

The runtime expression is defined by the following [ABNF](#) syntax

```

expression = ( "$url" | "$method" | "$statusCode" | "$request." source |
"$response." source )
source = ( header-reference | query-reference | path-reference | body-
reference )
header-reference = "header." token
query-reference = "query." name
path-reference = "path." name
body-reference = "body" [ "#" fragment ]
fragment = a JSON Pointer [RFC 6901](https://tools.ietf.org/html/rfc6901)
name = *( char )
char = as per RFC [7159](https://tools.ietf.org/html/rfc7159#section-7)
token = as per RFC [7230](https://tools.ietf.org/html/rfc7230#section-3.2.6)

```

The name identifier is case-sensitive, whereas token is not.

The table below provides examples of runtime expressions and examples of their use in a value:

Examples

Source Location	example expression	notes
HTTP Method	\$method	The allowable values for the \$method will be those for the HTTP operation.

Source Location	example expression	notes
Requested media type	<code>\$request.header.accept</code>	
Request parameter	<code>\$request.path.id</code>	Request parameters MUST be declared in the parameters section of the parent operation or they cannot be evaluated. This includes request headers.
Request body property	<code>\$request.body#/user/uuid</code>	In operations which accept payloads, references may be made to portions of the requestBody or the entire body.
Request URL	<code>\$url</code>	
Response value	<code>\$response.body#/status</code>	In operations which return payloads, references may be made to portions of the response body or the entire body.
Response header	<code>\$response.header.Server</code>	Single header values only are available

Runtime expressions preserve the type of the referenced value. Expressions can be embedded into string values by surrounding the expression with `{ }` curly braces.

Header Object

The Header Object follows the structure of the [Parameter Object](#) with the following changes:

1. name MUST NOT be specified, it is given in the corresponding headers map.
2. in MUST NOT be specified, it is implicitly in header.
3. All traits that are affected by the location MUST be applicable to a location of header (for example, [style](#)).

Header Object Example

A simple header of type integer:

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

```

    }
  }
  description: The number of allowed requests in the current period
  schema:
    type: integer

```

Tag Object

Adds metadata to a single tag that is used by the [Operation Object](#). It is not mandatory to have a Tag Object per tag defined in the Operation Object instances.

Fixed Fields

Field Name	Type	Description
name	string	REQUIRED. The name of the tag.
description	string	A short description for the tag. CommonMark syntax MAY be used for rich text representation.
externalDocs	External Documentation Object	Additional external documentation for this tag.

This object MAY be extended with [Specification Extensions](#).

Tag Object Example

```

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

```

Reference Object

A simple object to allow referencing other components in the specification, internally and externally.

The Reference Object is defined by [JSON Reference](#) and follows the same structure, behavior and rules.

For this specification, reference resolution is accomplished as defined by the JSON Reference specification and not by the JSON Schema specification.

Fixed Fields

Field Name	Type	Description
\$ref	string	REQUIRED. The reference string.

This object cannot be extended with additional properties and any properties added SHALL be ignored.

Reference Object Example

```
{
  "$ref": "#/components/schemas/Pet"
}
$ref: '#/components/schemas/Pet'
```

Relative Schema Document Example

```
{
  "$ref": "Pet.json"
}
$ref: Pet.yaml
```

Relative Documents With Embedded Schema Example

```
{
  "$ref": "definitions.json#/Pet"
}
$ref: definitions.yaml#/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 an extended subset of the [JSON Schema Specification Wright Draft 00](#).

For more information about the properties, see [JSON Schema Core](#) and [JSON Schema Validation](#). Unless stated otherwise, the property definitions follow the JSON Schema.

Properties

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

- title
- multipleOf
- maximum
- exclusiveMaximum

- minimum
- exclusiveMinimum
- maxLength
- minLength
- pattern (This string SHOULD be a valid regular expression, according to the [ECMA 262 regular expression](#) dialect)
- maxItems
- minItems
- uniqueItems
- maxProperties
- minProperties
- required
- enum

The following properties are taken from the JSON Schema definition but their definitions were adjusted to the OpenAPI Specification.

- type - Value MUST be a string. Multiple types via an array are not supported.
- allOf - Inline or referenced schema MUST be of a [Schema Object](#) and not a standard JSON Schema.
- oneOf - Inline or referenced schema MUST be of a [Schema Object](#) and not a standard JSON Schema.
- anyOf - Inline or referenced schema MUST be of a [Schema Object](#) and not a standard JSON Schema.
- not - Inline or referenced schema MUST be of a [Schema Object](#) and not a standard JSON Schema.
- items - Value MUST be an object and not an array. Inline or referenced schema MUST be of a [Schema Object](#) and not a standard JSON Schema. items MUST be present if the type is array.
- properties - Property definitions MUST be a [Schema Object](#) and not a standard JSON Schema (inline or referenced).
- additionalProperties - Value can be boolean or object. Inline or referenced schema MUST be of a [Schema Object](#) and not a standard JSON Schema. Consistent with JSON Schema, additionalProperties defaults to true.
- description - [CommonMark syntax](#) MAY be used for rich text representation.
- format - See [Data Type Formats](#) for further details. While relying on JSON Schema's defined formats, the OAS offers a few additional predefined formats.
- default - The default value represents what would be assumed by the consumer of the input as the value of the schema if one is not provided. Unlike JSON Schema, the value MUST conform to the defined type for the Schema Object defined at the same level. For example, if type is string, then default can be "foo" but cannot be 1.

Alternatively, any time a Schema Object can be used, a [Reference Object](#) can be used in its place. This allows referencing definitions instead of defining them inline.

Additional properties defined by the JSON Schema specification that are not mentioned here are strictly unsupported.

Other than the JSON Schema subset fields, the following fields MAY be used for further schema documentation:

Fixed Fields

Field Name	Type	Description
nullable	boolean	Allows sending a null value for the defined schema. Default value is false.
discriminator	Discriminator Object	Adds support for polymorphism. The discriminator is an object name that is used to differentiate between other schemas which may satisfy the payload description. See Composition and Inheritance for more details.
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 SHOULD NOT be sent as part of the request. If the property is marked as readOnly being true and is in the required list, the required will take effect on the response only. A property MUST NOT be marked as both readOnly and writeOnly being true. Default value is false.
writeOnly	boolean	Relevant only for Schema "properties" definitions. Declares the property as "write only". Therefore, it MAY be sent as part of a request but SHOULD NOT be sent as part of the response. If the property is marked as writeOnly being true and is in the required list, the required will take effect on the request only. A property MUST NOT be marked as both readOnly and writeOnly being true. 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 of this property.

Field Name	Type	Description
externalDocs	External Documentation Object	Additional external documentation for this schema.
example	Any	A free-form property to include an example of an instance for this schema. To represent examples that cannot be naturally represented in JSON or YAML, a string value can be used to contain the example with escaping where necessary.
deprecated	boolean	Specifies that a schema is deprecated and SHOULD be transitioned out of usage. Default value is false.

This object MAY be extended with [Specification Extensions](#).

Composition and Inheritance (Polymorphism)

The OpenAPI Specification allows combining and extending model definitions using the `allOf` property of JSON Schema, in effect offering model composition. `allOf` takes 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, the OpenAPI Specification adds the `discriminator` field. When used, the `discriminator` will be the name of the property that decides which schema definition validates the structure of the model. As such, the `discriminator` field MUST be a required field. There are two ways to define the value of a discriminator for an inheriting instance.

- Use the schema name.
- Override the schema name by overriding the property with a new value. If a new value exists, this takes precedence over the schema name. 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

```
{
```

```

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

```

Simple Model

```

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

```

Model with Map/Dictionary Properties

For a simple string to string mapping:

```

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

```

For a string to model mapping:

```

{
  "type": "object",

```

```

    "additionalProperties": {
      "$ref": "#/components/schemas/ComplexModel"
    }
  }
  type: object
  additionalProperties:
    $ref: '#/components/schemas/ComplexModel'

```

Model with Example

```

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

```

Models with Composition

```

{
  "components": {
    "schemas": {
      "ErrorModel": {
        "type": "object",
        "required": [
          "message",
          "code"
        ],
        "properties": {
          "message": {
            "type": "string"
          },
          "code": {

```

```

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

```

Models with Polymorphism Support

```

{
  "components": {
    "schemas": {
      "Pet": {
        "type": "object",

```

```

    "discriminator": {
      "propertyName": "petType"
    },
    "properties": {
      "name": {
        "type": "string"
      },
      "petType": {
        "type": "string"
      }
    },
    "required": [
      "name",
      "petType"
    ]
  },
  "Cat": {
    "description": "A representation of a cat. Note that `Cat` will be used as
the discriminator value.",
    "allOf": [
      {
        "$ref": "#/components/schemas/Pet"
      },
      {
        "type": "object",
        "properties": {
          "huntingSkill": {
            "type": "string",
            "description": "The measured skill for hunting",
            "default": "lazy",
            "enum": [
              "clueless",
              "lazy",
              "adventurous",
              "aggressive"
            ]
          }
        },
        "required": [
          "huntingSkill"
        ]
      }
    ]
  },
  "Dog": {
    "description": "A representation of a dog. Note that `Dog` will be used as
the discriminator value.",
    "allOf": [
      {
        "$ref": "#/components/schemas/Pet"
      },
      {
        "type": "object",
        "properties": {
          "packSize": {
            "type": "integer",
            "format": "int32",
            "description": "the size of the pack the dog is from",
            "default": 0,

```

```

        "minimum": 0
      }
    },
    "required": [
      "packSize"
    ]
  }
]
}
}
}
}
}
components:
  schemas:
    Pet:
      type: object
      discriminator:
        propertyName: petType
      properties:
        name:
          type: string
        petType:
          type: string
      required:
        - name
        - petType
    Cat: ## "Cat" will be used as the discriminator value
      description: A representation of a cat
      allOf:
        - $ref: '#/components/schemas/Pet'
        - type: object
          properties:
            huntingSkill:
              type: string
              description: The measured skill for hunting
              enum:
                - clueless
                - lazy
                - adventurous
                - aggressive
              required:
                - huntingSkill
    Dog: ## "Dog" will be used as the discriminator value
      description: A representation of a dog
      allOf:
        - $ref: '#/components/schemas/Pet'
        - type: object
          properties:
            packSize:
              type: integer
              format: int32
              description: the size of the pack the dog is from
              default: 0
              minimum: 0
            required:
              - packSize

```

Discriminator Object

When request bodies or response payloads may be one of a number of different schemas, a discriminator object can be used to aid in serialization, deserialization, and validation. The discriminator is a specific object in a schema which is used to inform the consumer of the specification of an alternative schema based on the value associated with it.

When using the discriminator, *inline* schemas will not be considered.

Fixed Fields

Field Name	Type	Description
propertyName	string	REQUIRED. The name of the property in the payload that will hold the discriminator value.
mapping	Map[string, string]	An object to hold mappings between payload values and schema names or references.

The discriminator object is legal only when using one of the composite keywords `oneOf`, `anyOf`, `allOf`.

In OAS 3.0, a response payload MAY be described to be exactly one of any number of types:

```
MyResponseType:
  oneOf:
    - $ref: '#/components/schemas/Cat'
    - $ref: '#/components/schemas/Dog'
    - $ref: '#/components/schemas/Lizard'
```

which means the payload *MUST*, by validation, match exactly one of the schemas described by Cat, Dog, or Lizard. In this case, a discriminator MAY act as a "hint" to shortcut validation and selection of the matching schema which may be a costly operation, depending on the complexity of the schema. We can then describe exactly which field tells us which schema to use:

```
MyResponseType:
  oneOf:
    - $ref: '#/components/schemas/Cat'
    - $ref: '#/components/schemas/Dog'
    - $ref: '#/components/schemas/Lizard'
  discriminator:
    propertyName: petType
```

The expectation now is that a property with name `petType` *MUST* be present in the response payload, and the value will correspond to the name of a schema defined in the OAS document. Thus the response payload:

```
{
  "id": 12345,
  "petType": "Cat"
}
```

Will indicate that the cat schema be used in conjunction with this payload.

In scenarios where the value of the discriminator field does not match the schema name or implicit mapping is not possible, an optional mapping definition MAY be used:

```
MyResponseType:
  oneOf:
    - $ref: '#/components/schemas/Cat'
    - $ref: '#/components/schemas/Dog'
    - $ref: '#/components/schemas/Lizard'
    - $ref: 'https://gigantic-server.com/schemas/Monster/schema.json'
  discriminator:
    propertyName: petType
    mapping:
      dog: '#/components/schemas/Dog'
      monster: 'https://gigantic-server.com/schemas/Monster/schema.json'
```

Here the discriminator *value* of dog will map to the schema `#/components/schemas/Dog`, rather than the default (implicit) value of `Dog`. If the discriminator *value* does not match an implicit or explicit mapping, no schema can be determined and validation SHOULD fail. Mapping keys MUST be string values, but tooling MAY convert response values to strings for comparison.

When used in conjunction with the `anyOf` construct, the use of the discriminator can avoid ambiguity where multiple schemas may satisfy a single payload.

In both the `oneOf` and `anyOf` use cases, all possible schemas MUST be listed explicitly. To avoid redundancy, the discriminator MAY be added to a parent schema definition, and all schemas comprising the parent schema in an `allOf` construct may be used as an alternate schema.

For example:

```
components:
  schemas:
    Pet:
      type: object
      required:
        - petType
      properties:
        petType:
          type: string
      discriminator:
        propertyName: petType
        mapping:
          dog: Dog
    Cat:
      allOf:
        - $ref: '#/components/schemas/Pet'
        - type: object
          # all other properties specific to a `Cat`
          properties:
            name:
              type: string
    Dog:
      allOf:
        - $ref: '#/components/schemas/Pet'
        - type: object
          # all other properties specific to a `Dog`
          properties:
```

```

      bark:
        type: string
    Lizard:
      allOf:
        - $ref: '#/components/schemas/Pet'
        - type: object
          # all other properties specific to a `Lizard`
          properties:
            lovesRocks:
              type: boolean

```

a payload like this:

```

{
  "petType": "Cat",
  "name": "misty"
}

```

will indicate that the cat schema be used. Likewise this schema:

```

{
  "petType": "dog",
  "bark": "soft"
}

```

will map to Dog because of the definition in the mappings element.

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 <code>items</code> , it will affect the name of the individual XML elements within the list. When defined alongside <code>type</code> being array (outside the <code>items</code>), it will affect the wrapping element and only if <code>wrapped</code> is true. If <code>wrapped</code> is false, it will be ignored.
namespace	string	The URI of the namespace definition. Value MUST be in the form of an absolute URI.
prefix	string	The prefix to be used for the name .

Field Name	Type	Description
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).

This object MAY be extended with [Specification Extensions](#).

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://example.com/schema/sample",
          "prefix": "sample"
        }
      }
    }
  }
}
}
}
Person:
  type: object
  properties:
    id:
      type: integer
      format: int32
      xml:
        attribute: true
    name:
      type: string
      xml:
        namespace: http://example.com/schema/sample
        prefix: sample
<Person id="123">
  <sample:name
xmlns:sample="http://example.com/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 a name is not 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 naming problem in the example above, the following definition can be used:

```

{
  "animals": {
    "type": "array",
    "items": {
      "type": "string",
      "xml": {
        "name": "animal"
      }
    },
    "xml": {
      "wrapped": true
    }
  }
}
animals:
  type: array
  items:
    type: string
  xml:
    name: animal
  xml:
    wrapped: true
<animals>
  <animal>value</animal>
  <animal>value</animal>
</animals>

```

Affecting both internal and external names:

```

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

```

```

animals:
  type: array
  items:
    type: string
    xml:
      name: animal
  xml:
    name: aliens
    wrapped: true
<aliens>
  <animal>value</animal>
  <animal>value</animal>
</aliens>

```

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

```

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

```

```

animals:
  type: array
  items:
    type: string
  xml:
    name: aliens
    wrapped: true
<aliens>
  <aliens>value</aliens>
  <aliens>value</aliens>
</aliens>

```

Security Scheme Object

Defines a security scheme that can be used by the operations. Supported schemes are HTTP authentication, an API key (either as a header, a cookie parameter or as a query parameter), OAuth2's common flows (implicit, password, application and access code) as defined in [RFC6749](#), and [OpenID Connect Discovery](#).

Fixed Fields

Field Name	Type	Applies To	Description
type	string	Any	REQUIRED. The type of the security scheme. Valid values are "apiKey", "http", "oauth2", "openIdConnect".
description	string	Any	A short description for security scheme. CommonMark syntax MAY be used for rich text representation.
name	string	apiKey	REQUIRED. The name of the header, query or cookie parameter to be used.
in	string	apiKey	REQUIRED. The location of the API key. Valid values are "query", "header" or "cookie".
scheme	string	http	REQUIRED. The name of the HTTP Authorization scheme to be used in the Authorization header as defined in RFC7235 .
bearerFormat	string	http ("bearer")	A hint to the client to identify how the bearer token is formatted. Bearer tokens are usually generated by an authorization server, so this information is primarily for documentation purposes.
flows	OAuth Flows Object	oauth2	REQUIRED. An object containing configuration information for the flow types supported.
openIdConnectUrl	string	openIdConnect	REQUIRED. OpenId Connect URL to discover OAuth2 configuration values. This MUST be in the form of a URL.

This object MAY be extended with [Specification Extensions](#).

Security Scheme Object Example

Basic Authentication Sample

```
{
  "type": "http",
  "scheme": "basic"
}
```



```
type: http
scheme: basic
```

API Key Sample

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

JWT Bearer Sample

```
{
  "type": "http",
  "scheme": "bearer",
  "bearerFormat": "JWT",
}
type: http
scheme: bearer
bearerFormat: JWT
```

Implicit OAuth2 Sample

```
{
  "type": "oauth2",
  "flows": {
    "implicit": {
      "authorizationUrl": "https://example.com/api/oauth/dialog",
      "scopes": {
        "write:pets": "modify pets in your account",
        "read:pets": "read your pets"
      }
    }
  }
}
type: oauth2
flows:
  implicit:
    authorizationUrl: https://example.com/api/oauth/dialog
    scopes:
      write:pets: modify pets in your account
      read:pets: read your pets
```

OAuth Flows Object

Allows configuration of the supported OAuth Flows.

Fixed Fields

Field Name	Type	Description
implicit	OAuth Flow Object	Configuration for the OAuth Implicit flow
password	OAuth Flow Object	Configuration for the OAuth Resource Owner Password flow
clientCredentials	OAuth Flow Object	Configuration for the OAuth Client Credentials flow. Previously called application in OpenAPI 2.0.
authorizationCode	OAuth Flow Object	Configuration for the OAuth Authorization Code flow. Previously called accessCode in OpenAPI 2.0.

This object MAY be extended with [Specification Extensions](#).

OAuth Flow Object

Configuration details for a supported OAuth Flow

Fixed Fields

Field Name	Type	Applies To	Description
authorizationUrl	string	oauth2 ("implicit", "authorizationCode")	REQUIRED. The authorization URL to be used for this flow. This MUST be in the form of a URL.

Field Name	Type	Applies To	Description
tokenUrl	string	oauth2 ("password", "clientCredentials", "authorizationCode")	REQUIRED. The token URL to be used for this flow. This MUST be in the form of a URL.
refreshUrl	string	oauth2	The URL to be used for obtaining refresh tokens. This MUST be in the form of a URL.
scopes	Map[string, string]	oauth2	REQUIRED. The available scopes for the OAuth2 security scheme. A map between the

Field Name	Type	Applies To	Description
			scope name and a short description for it.

This object MAY be extended with [Specification Extensions](#).

OAuth Flow Object Examples

```
{
  "type": "oauth2",
  "flows": {
    "implicit": {
      "authorizationUrl": "https://example.com/api/oauth/dialog",
      "scopes": {
        "write:pets": "modify pets in your account",
        "read:pets": "read your pets"
      }
    },
    "authorizationCode": {
      "authorizationUrl": "https://example.com/api/oauth/dialog",
      "tokenUrl": "https://example.com/api/oauth/token",
      "scopes": {
        "write:pets": "modify pets in your account",
        "read:pets": "read your pets"
      }
    }
  }
}
```

```
type: oauth2
flows:
  implicit:
    authorizationUrl: https://example.com/api/oauth/dialog
    scopes:
      write:pets: modify pets in your account
      read:pets: read your pets
  authorizationCode:
    authorizationUrl: https://example.com/api/oauth/dialog
    tokenUrl: https://example.com/api/oauth/token
    scopes:
      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 name used for each property MUST correspond to a security scheme declared in the [Security Schemes](#) under the [Components Object](#).

Security Requirement Objects that contain multiple schemes require that all schemes MUST be satisfied for a request to be authorized. This enables support for scenarios where multiple query parameters or HTTP headers are required to convey security information.

When a list of Security Requirement Objects is defined on the [OpenAPI Object](#) or [Operation Object](#), only one of the Security Requirement Objects in the list needs to be satisfied to authorize the request.

Patterned Fields

Field Pattern	Type	Description
{name}	[string]	Each name MUST correspond to a security scheme which is declared in the Security Schemes under the Components Object . If the security scheme is of type "oauth2" or "openIdConnect", 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 OpenAPI Specification tries to accommodate most use cases, additional data can be added to extend the specification at certain points.

The extensions properties are implemented as patterned fields that are always prefixed by "x-".

Field Pattern	Type	Description
[^] x-	Any	Allows extensions to the OpenAPI 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. 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 OpenAPI Specification MAY be declared and remain empty, or be completely removed, even though they are inherently the core of the API documentation.

The reasoning is to allow an additional layer of access control over the documentation. 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 of 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 to finely control what the viewer can see.