

附錄 2

OpenAPI Specification 中文摘要譯本

Version 3.0.0-rc2

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 RFC2119RFC8174](#) when, and only when, they appear in all capitals, as shown here.

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

目 錄

壹、導論.....	1
貳、定義.....	1
一、OpenAPI 定義文件 (OpenAPI Definition File).....	1
二、路徑模板 (Path Templating).....	1
三、媒體類型 (Mime Types).....	1
四、HTTP 狀態碼 (HTTP Status Codes).....	2
參、規格.....	2
一、格式 (Format).....	2
二、文件結構 (File Structure).....	2
三、資料類型 (Data Types).....	3
四、富文字格式 (Rich Text Formatting).....	3
五、URL 相關參考 (Relative References in URLs).....	4
六、結構描述 (Schema).....	4
1.OpenAPI 物件 (OpenAPI Object).....	4
2.資訊物件 (Info Object).....	5
3.聯絡物件 (Contact Object).....	6
4.授權物件 (License Object).....	7
5.伺服器物件 (Server Object).....	7
6.伺服器變數物件 (Server Variable Object).....	10
7.元件物件 (Components Object).....	10
8.路徑物件 (Paths Object).....	15
9.路徑項目物件 (Path Item Object).....	16
10.操作物件 (Operation Object).....	19
11.外部文件物件 (External Documentation Object).....	23
12.參數物件 (Parameter Object).....	24
13.Request Body 物件 (Request Body Object).....	30

14.媒體類型物件 (Media Type Object).....	33
15.編碼物件 (Encoding Object).....	38
16.(複)回應物件 (Responses Object).....	39
17.單一回應物件 (Response Object).....	41
18.回呼物件 (Callback Object).....	44
19.範例物件 (Example Object).....	46
20.連結物件 (Link Object).....	48
21.標頭物件 (Header Object).....	59
22.標籤物件 (Tag Object).....	59
23.(複)範例物件 (Examples Object).....	60
24.參考物件 (Reference Object).....	61
25.Schema 物件 (Schema Object).....	62
26.識別物件 (Discriminator Object).....	73
27.XML 物件 (XML Object).....	77
28.安全方案物件 (Security Scheme Object).....	84
29.OAuth 流程物件 (OAuth Flows Object).....	86
30.OAuth 單一流程物件 (OAuth Flow Object).....	86
31.安全需求物件 (Security Requirement Object).....	88
七、規格擴充 (Specification Extensions).....	89
八、安全篩選 (Security Filtering).....	89
附錄 A: 歷史版本.....	90

壹、導論

OpenAPI Specification (OAS)是用於描述 RESTful APIs 的標準。

OpenAPI Specification 定義了一組描述 API 所需之文件，前揭文件可採用文件產生工具來展現 API，並可用編碼產生工具以建置各種程式語言之本地服務。

亦可藉由前揭文件產生額外應用，例如測試工具等。

貳、定義

一、OpenAPI 定義文件 (OpenAPI Definition File)

一份定義 API 的文件，該文件符合 OpenAPI Specification 標準。

二、路徑模板 (Path Templating)

路徑模板係使用大括弧({ })來標記一組 URL，作為可更換的路徑參數。

三、媒體類型 (Mime Types)

媒體類型定義分散於多種資源中。

媒體類型定義應該符合 [RFC 6838](#)，以下為可能是媒體類型定義的範例：

```
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
```

四、HTTP 狀態碼 (HTTP Status Codes)

HTTP 狀態碼用於指涉被執行操作之狀態。可使用的狀態碼由 [RFC 7231](#) 所定義並且已註冊狀態碼亦已列示於 [IANA Status Code Registry](#)。

參、規格

一、格式 (Format)

依據本標準描述 RESTful API 的文件是以多個 JSON 物件構成，並且符合 JSON 標準。作為 JSON 格式的母集合，亦可以 YAML 格式呈現一份 OAS 文件。

例如，若一組欄位為一陣列值，則可採 JSON 陣列表示：

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

所有在本規範規格中的欄位名稱皆是大小寫有別。

schema 揭露了兩中欄位：一是作為宣告名稱的「固定欄位」，以及針對欄位名稱宣告一正規表示式的「態樣欄位」。

態樣欄位在一 containing object 內必須有唯一的名稱。

為了維持 YAML 與 JSON 格式與內容的互通性，建議採用 YAML version 1.2 版以及一些額外限制：

- Tags 必須限制在 [JSON Schema ruleset](#) 所允許的範圍。
- YAML 映射用的 key 值必須為純量字串，並符合 [YAML Failsafe schema ruleset](#) 所定義。

注意：雖然 API 是由 YAML 或 JSON 格式的 OAS 標準所描述，API 本身的輸入和回傳值以及其他內容則不需要是 JSON 或 YAML。

二、文件結構 (File Structure)

API 的 OAS 文件由單一檔案製成。然而，部分定義可以在使用者斟酌下分列於不同的文件中。這點可以適用於規格中的 \$ref 欄位，遵循 [JSON Schema](#) 的定義。按照慣例，建議將 OAS 文件命名為 openapi.json 或

openapi.yaml。

三、資料類型 (Data Types)

在 OAS 中的基本資料類型是依據 [JSON Schema Specification Wright Draft 00](#) 支援的類型訂定。注意當類型被指定為 integer 時，其等同於 JSON 的 number 定義，而不包含分數或指數。其類別(Type)不能設定為 null。模型使用 JSON Schema Specification Wright Draft 00 的一擴充子集 [Schema Object](#) 來描述。

format 為一個基本的、可選填的、可修改的屬性。OAS 使用幾種已知的格式來強化定義已使用的資料類型。然而，為了支援各類的文件需求，format 具有開放式的字串屬性，可以填入任何的值，例如"email"、"uuid"等等本標準未定義的值。不具備 format 屬性之資料類型須遵照其在 JSON Schema 的定義。無法被工具所識別的 format 將單獨返回預設資料類型，猶如該 format 未被特別指定。根據 OAS 定義的格式為：

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

四、富文字格式 (Rich Text Formatting)

在這整份定義中，description 欄位被註明為可標記為支援 CommonMark (一種輕量級標記式語言)。OpenAPI 工具必須支援輕量級標記式語言

[CommonMark 0.27](#) 以呈現富文字。工具可以選擇忽略一些 CommonMark 功能以排除安全性問題。

五、URL 相關參考 (Relative References in URLs)

除非特定情況，所有 URL 的屬性如 [RFC 3986](#) 之定義，可以作為相關參考。利用於 [Server Object](#) 中定義的 URL 作為基礎 URI，相關參考是可以被解決的。

在 \$ref 中利用的相關參考要如每一筆 [JSON Reference](#) 來被處理，例如目前文件中正在作為基礎 URI 使用的 URL。同時請參考 [Reference Object](#)。

六、結構描述 (Schema)

在下列描述中，如果某一欄位沒有明確標明**必要(REQUIRED)**，或以**必須(MUST)**、**應該(SHALL)**所描述的，都視為可選填。

1. OpenAPI 物件 (OpenAPI Object)

這是一份 [OpenAPI definition file](#) 的基礎物件。

Fixed Fields

Field Name	Type	Description
openapi	string	必填 ，這字串必須載明該文件所使用的 OpenAPI Specification 版本。這個 openapi 欄位 應該 要讓使用者可藉由工具直譯其版本。該欄位與 API info 版本號無關。
info	Info Object	必填 ，提供這份 API 的詮釋資料。如有需要，這詮釋資料亦可由使用者所使用。
servers	[Server Object]	伺服器物件，可提供至目標伺服器之連結資訊。若未提供該欄位，或為空陣列，則伺服器物件欄位之預設 URL 將會是根目錄"/"。
paths	Paths Object	必填 ，記載這份 API 的功能操作及可用路徑。
components	Components Object	用於記載保存於各種 schema 之元素。
security	[Security	宣告其可跨用於整份 API 之安全機制。其清單內包含可供使

Field Name	Type	Description
	Requirement Object	用的 security requirement objects。僅需有一項 security requirement objects 滿足授權需求即可。可藉由個別操作覆蓋此定義。
tags	[Tag Object]	本標準於附加詮釋資料所使用的標籤清單。標籤順序可被分析工具所解析。並非所有 Operation Object 所使用的標籤都必須被宣告，未被宣告之標籤可被隨機地或邏輯性地組織起來。清單中每個標籤名稱都必得是獨一無二的。
externalDocs	External Documentation Object	Additional external documentation.

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

2. 資訊物件 (Info Object)

The object provides metadata about the API. The metadata can be used by the clients if needed, and can 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 can be used for rich text representation.
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 API definition (which is distinct from the OpenAPI specification version or the API implementation version).

This object can 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": "http://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: http://www.apache.org/licenses/LICENSE-2.0.html
version: 1.0.1

```

3. 聯絡物件 (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 can 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

```

4. 授權物件 (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 can be extended with [Specification Extensions](#).

License Object Example:

```

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

```

5. 伺服器物件 (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 definition 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 can 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 can 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`

6. 伺服器變數物件 (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 if an alternate value is not specified, and will be sent if an alternative value is <i>not</i> supplied. Unlike the Schema Object's <code>default</code> , this value MUST be provided by the consumer.
description	string	An optional description for the server variable. CommonMark syntax can be used for rich text representation.

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

7. 元件物件 (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 .

Field Name	Type	Description
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 can 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": {
    "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:
    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

```

8. 路徑物件 (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
<code>/path</code>	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 <code>url</code> field in order to construct the full URL. Path templating is allowed.

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

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.",

```


Field Name	Type	Description
		conflicts between the referenced definition and this Path Item's definition, the behavior is <i>undefined</i> .
summary	string	An optional, string summary, intended to apply to all operations in this path.
description	string	An optional, string description, intended to apply to all operations in this path. CommonMark syntax can be used for rich text representation.
get	Operation Object	A definition of a GET operation on this path.
put	Operation Object	A definition of a PUT operation on this path.
post	Operation Object	A definition of a POST operation on this path.
delete	Operation Object	A definition of a DELETE operation on this path.
options	Operation Object	A definition of a OPTIONS operation on this path.
head	Operation Object	A definition of a HEAD operation on this path.
patch	Operation Object	A definition of a PATCH operation on this path.
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 can 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

```

10. 操作物件 (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.

Field Name	Type	Description
description	string	A verbose explanation of the operation behavior. CommonMark syntax can 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. 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 .
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 can 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,
      "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": "Invalid input",
      "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
  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: Invalid input
    content:
      'application/json': {}
      'application/xml': {}
security:
  - petstore_auth:
    - write:pets
    - read:pets

```

11. 外部文件物件 (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 can be used for rich text representation.
url	string	REQUIRED. The URL for the target documentation. Value MUST be in the format of a URL.

This object can 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

```

12. 參數物件 (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 (as specified with 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	REQUIRED. The name of the parameter. Parameter names are <i>case sensitive</i>. If <code>in</code> is "path", the name field MUST correspond to the associated path segment from the <code>path</code> field in the Paths Object. See Path Templating for further information. If <code>in</code> 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 based on the <code>in</code> property.
in	string	REQUIRED. The location of the parameter. Possible values are "query", "header", "path" or "cookie".
description	string	A brief description of the parameter. This could contain examples of use. CommonMark syntax can be used for rich text representation.

Field Name	Type	Description
required	boolean	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.
deprecated	boolean	Specifies that a parameter is deprecated and SHOULD be transitioned out of usage.
allowEmptyValue	boolean	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, if behavior is n/a, the value of allowEmptyValue SHALL be ignored.

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

Field Name	Type	Description
style	string	Describes how the parameter value will be serialized depending on 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 object is mutually exclusive to the examples object. Furthermore, if referencing a schema which contains an example, the example value SHALL <i>override</i> the 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 be used to 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 object is mutually exclusive to

Field Name	Type	Description
	Example Object	the example object. Furthermore, if referencing a schema which contains an example, the examples value SHALL <i>override</i> the example provided by the schema.

For more complex scenarios the [content](#) property can be used to 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 is used to describe 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.
simple	array	path, header	Simple style parameters defined by RFC6570 . This option replaces <code>collectionFormat</code> with a <code>csv</code> value from OpenAPI 2.0.
spaceDelimited	array	query	Space separated array values. This option replaces <code>collectionFormat</code> equal to <code>ssv</code> from OpenAPI 2.0.
pipeDelimited	array	query	Pipe separated array values. This option replaces <code>collectionFormat</code> equal to <code>pipes</code> from OpenAPI 2.0.
deepObject	object	query	Provides a simple way of rendering nested objects using form parameters.

Style Examples

Assuming a parameter named color with one of the following values:

string -> "blue"

array -> ["blue","black","brown"]

object -> { "R": 100, "G": 200, "B": 150 }

The following table shows examples of how those values would be rendered.

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
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 can 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
```

13. Request Body 物件 (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 can be used for rich text representation.
content	Map[string, Media Type Object]	REQUIRED. The content of the request body. The key is the media type and the value is used to describe it.
required	boolean	Determines if the request body is required in the request. Defaults to false.

This object can 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

```

14. 媒體類型物件 (Media Type Object)

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

Fixed Fields

Field Name	Type	Description
schema	Schema Object Reference Object	The schema defining the type used for the request body.
example	Any	Example of the media type. The example object SHOULD be in the correct format as specified in the media type. The example object is mutually exclusive to the examplesobject. Furthermore, if referencing a schema which contains an example, the examplevalue SHALL <i>override</i> the 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 object is mutually exclusive to the exampleobject. Furthermore, if referencing a schema which contains an example, the examplesvalue 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 SHOULD only apply to requestBody objects when the content type is multipart.

This object can 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, describing 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
```

Note that the above examples apply to either input payloads (i.e. file uploads) or response payloads.

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

```
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:
    'image/png, image/jpeg':
      # a binary file of type png or jpeg
      schema:
```

```
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:
    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: {}
```

Note that in the above example, the contents in the requestBody MUST be stringified per [RFC1866](#) when being passed to the server. In addition, the address field complex object will be stringified as well.

When passing complex objects in the x-www-form-urlencoded content type, the default serialization strategy of such properties is described in the parameterContent section 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 allows 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'
```

In scenarios where more control is needed over the Content-Type for multipart request bodies, an encoding attribute is introduced. This attribute

is only applicable to multipart/* and x-www-form-urlencoded request bodies.

15. 編碼物件 (Encoding Object)

A single encoding definition applied to a single schema property.

Fixed Fields

Field Name	Type	Description
contentType	string	The Content-Type to use 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.
headers	Map[string, Header Object]	A string map allowing additional information to be provided as headers, for example Content-Disposition. Note Content-Type is described separately and will be ignored from this section.
style	string	Describes how a specific property value will be serialized depending on its type . See Parameter Object for details on the style property. The behavior follows the same values allowed for query parameters, including default values.
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.
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 object can 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
      type: integer

```

16. (複)回應物件 (Responses Object)

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

It is not expected for the documentation to necessarily cover all possible HTTP response codes, since they may not be known in advance. However, it is expected for the documentation 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. It can be used to cover undeclared responses. Reference Object can be used to link to a response that is defined at the OpenAPI Object's components/responses section.

Patterned Fields

Field Pattern	Type	Description
HTTP Status Code	Response Object Reference Object	Any HTTP status code can be used as the property name (one property per HTTP status code). Describes the expected response for that HTTP status code. Reference Object can be used to link to a response that is defined at the OpenAPI Object's components/responses section. This field MUST be quoted for compatibility between JSON and YAML (i.e. "200"), and MAY contain the uppercase character, X to designate a wildcard, such as 2XX to represent all response codes between [200-299]. If a response range is defined, and an explicit code within that range is defined as well, the explicit code definition takes precedence over the range definition for that code.

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

Responses Object Example

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

```
{
  "200": {
    "description": "a pet to be returned",
    "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/ErrorMessage'

```

17. 單一回應物件 (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 can be used for rich text representation.
headers	Map[string, Header Object Reference Object]	Maps a header name to its definition. Note that 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 the media type and the value is used to describe it.
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

Field Name	Type	Description
		for Component Objects .

This object can 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
representations:
  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

```

18. 回呼物件 (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

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

Key Expression

The key used to identify the [Path Item Object](#) is a variable 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 [variable substitution](#) syntax the complete HTTP message can be accessed. This includes accessing any part of a body that can be accessed using a JSON Pointer [RFC6901](#).

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: 123
```

```
{
  "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>

Here are the examples of how the various expressions evaluate, assuming a 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
<code>\$request.header.content-Type</code>	application/json
<code>\$request.body#/failedUrl</code>	http://clientdomain.com/stillrunning

Expression	Value
<code>\$request.body#/successUrls/2</code>	http://clientdomain.com/medium
<code>\$response.header.Location</code>	http://example.org/subscription/1

Callback Object Example

A callback to the URL specified by the url parameter in the request

```
myWebhook:
  '$request.body#/url':
    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
```

19. 範例物件 (Example Object)

Fixed Fields

Field Name	Type	Description
summary	string	Short description for the example.
description	string	Long description for the example. CommonMark syntax can 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, a string value can be used to contain the example with escaping where necessary.
externalValue	string	A URL that points to the literal example. This provides the ability to reference examples that cannot easily be included in JSON or YAML documents. The value field and externalValue field are mutually exclusive.

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

In all cases, the example value is expected to be compatible with the type schema for the value that it is accompanying. Tooling implementations MAY choose to validate compatibility automatically, and reject the example value(s) if they are not

compatible.

Example Object Example

```
# 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'
```

20. 連結物件 (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.

As opposed to dynamic links (links provided **in** the response payload), the OAS linking mechanism does not require that link information be provided in a specific response format at runtime.

Many operations require parameters to be passed, and these MAY be dynamic depending on the response itself. For computing links, and providing instructions to execute them, [variable substitution](#) is used for accessing values in a response and using them as values 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 with the operationId field, and MUST point to the fragment of a valid OAS definition.
operationId	string	the name of an <i>existing</i> , resolvable OAS operation, as defined with a unique operationId. This field is mutually exclusive with the operationRef field. Relative operationRef values MAY be used to locate an existing Operation Object in the OAS.
parameters	Map[string Any {expression}]	A map representing parameters to pass to the 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.
headers	Map[string, Header Object Reference Object]	Maps a header name to its definition. Note that RFC7230 states header names are case insensitive. This represents the headers to pass to the linked resource. Where conflicts occur between these headers, and those defined in the related operation, these headers override.
description	string	a description of the link, supports CommonMark syntax .
server	Server Object	a server object to be used by the target operation.

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

Locating a linked resource MAY be performed by either a 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, consider the operationRef syntax as the preferred method for specifications with external references.

Response Payload Values

Payload values are only available in parsable response payloads which match the advertised media type and for media types that can be referenced using a JSON Pointer fragment Id. In all cases, if a value does not exist, the parameter will be considered a null value (as opposed to an empty value) and not passed as a parameter to the linked resource. In cases where a value is required, and a parameter is not supplied, the client MAY choose to not follow the link definition.

Example

Response payload:

```
{  
  "id": "df71a505-07bc-458a-a5c0-73c0340d1ec7",  
}
```

```

    "firstname": "Ash",
    "lastname": "Williams"
}

```

Payload Variables:

```

id: df71a505-07bc-458a-a5c0-73c0340d1ec7
firstname: Ash
lastname: Williams
missingValue: null

```

In situations where variables appear in an array, an array of variables will be extracted. For example:

```

[
  { "color": "red" },
  { "color": "green" },
  { "color": "blue" }
]

```

will be extracted as such:

```
color: ["red", "green", "blue"]
```

The variables generated can be used in locations prescribed by the definition.

Variable Substitution

In all cases, variables from request and responses MAY be substituted for link generation. The table below provides examples of variable expressions and examples of their use in a value:

Source Location	variable expression	example reference	notes
HTTP Method	<code>\$method</code>	<code>/users/{ \$method }</code>	The allowable values for the <code>\$method</code> will be those for the HTTP operation
Requested content type	<code>\$request.header.accept</code>	<code>/users/3?format={ \$request.header.accept }</code>	
Request parameter	<code>\$request.path.id</code>	<code>/users/{ \$request.path.id }</code>	Request parameters MUST be declared in the parameters section for the operation or they cannot be used in substitution. This includes request headers
Request body	<code>\$request.body</code>	<code>/users/{ \$request.body#/user/uid }</code>	For operations which accept payloads, references MAY be made to portions of

Source Location	variable expression	example reference	notes
			the requestBody or the entire body itself
Request URL	<code>\$url</code>	<code>/track?url={ \$url }</code>	
Response value	<code>\$response.body</code>	<code>{ \$response.body#/uuid }</code>	Only the payload in the response can be accessed with the <code>\$responsesyntax</code> .
Response header	<code>\$response.header</code>	<code>{ \$response.header.Server }</code>	Single header values only are available

From the request, the parameters used in calling the operation are made available through the `$request` syntax. For responses, the response payload MAY be used with the `$response` syntax. For both requests and responses, values will be substituted in the link in sections designated with a variable expression, surrounded by curly brackets `{ }`.

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

```

expression = ( "$url" | "$method" | "$request." [ source ] | "$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 [RFC6901](https://tools.ietf.org/html/rfc6901)
name = *( char )
char = as per [RFC7159](https://tools.ietf.org/html/rfc7159#section-7)
token = as per [RFC7230](https://tools.ietf.org/html/rfc7230#section-3.2.6)

```

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

Request Parameter Example

Computing a link from a request operation like this:

```

paths:
  /users/{id}:
    parameters:
      - name: id
        in: path

```

```

    required: true
    description: the user identifier, as userId or username
    schema:
      type: string
  responses:
    '200':
      description: the user being returned
      content:
        application/json:
          schema:
            type: object
            properties:
              uuid: the unique user id
              type: string
              format: uuid

```

Can be used in a link like this:

Addresses:

```

# the target link operationId
operationId: getUserAddress
parameters:
  # get the `id` field from the request path parameter named `id`
  userId: '{$request.path.id}'

```

Where the `$request.path.id` is the value passed in the request to `/users/{id}`.

Response Payload Example

Addresses:

```

operationId: getUserAddressByUUID
parameters:
  # get the `id` field from the request path parameter named `id`
  userUuid: '{$response.body#/uuid}'

```

And the array example:

ColorSelection:

```

operationId: getColorSample
parameters:
  colorName: '{$response.body#/color}'

```

Would produce three links with the `colorName` of red, green, and blue:

As with all links, it is at the clients' discretion to follow them, neither permissions nor the ability to make a successful call to that link is guaranteed solely by the existence of a relationship.

Example

The example below shows how relationships in the BitBucket API can be represented with the link schema. This example uses `operationId` values to link responses to possible operations.

```
paths:
  /2.0/users/{username}:
    get:
      operationId: getUserByName
      parameters:
        - name: username
          in: path
          required: true
          schema:
            type: string
      responses:
        '200':
          description: The User
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/user'
          links:
            userRepositories:
              $ref: '#/components/links/UserRepositories'
  /2.0/repositories/{username}:
    get:
      operationId: getRepositoriesByOwner
      parameters:
        - name: username
          in: path
          required: true
          schema:
            type: string
```

```

responses:
  '200':
    description: repositories owned by the supplied user
    content:
      application/json:
        schema:
          type: array
          items:
            $ref: '#/components/schemas/repository'
    links:
      userRepository:
        $ref: '#/components/links/UserRepository'
/2.0/repositories/{username}/{slug}:
get:
  operationId: getRepository
  parameters:
    - name: username
      in: path
      required: true
      schema:
        type: string
    - name: slug
      in: path
      required: true
      schema:
        type: string
  responses:
    '200':
      description: The repository
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/repository'
      links:
        repositoryPullRequests:
          $ref: '#/components/links/RepositoryPullRequests'
/2.0/repositories/{username}/{slug}/pullrequests:

```

```

get:
  operationId: getPullRequestsByRepository
  parameters:
    - name: username
      in: path
      required: true
      schema:
        type: string
    - name: slug
      in: path
      required: true
      schema:
        type: string
    - name: state
      in: query
      schema:
        type: string
        enum:
          - open
          - merged
          - declined
  responses:
    '200':
      description: an array of pull request objects
      content:
        application/json:
          schema:
            type: array
            items:
              $ref: '#/components/schemas/pullrequest'
/2.0/repositories/{username}/{slug}/pullrequests/{pid}:
get:
  operationId: getPullRequestsById
  parameters:
    - name: username
      in: path
      required: true

```

```

    schema:
      type: string
  - name: slug
    in: path
    required: true
    schema:
      type: string
  - name: pid
    in: path
    required: true
    schema:
      type: string
responses:
  '200':
    description: a pull request object
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/pullrequest'
    links:
      pullRequestMerge:
        $ref: '#/components/links/PullRequestMerge'
/2.0/repositories/{username}/{slug}/pullrequests/{pid}/merge:
  post:
    operationId: mergePullRequest
    parameters:
      - name: username
        in: path
        required: true
        schema:
          type: string
      - name: slug
        in: path
        required: true
        schema:
          type: string
      - name: pid

```

```

    in: path
    required: true
    schema:
      type: string
  responses:
    '204':
      description: the PR was successfully merged
components:
  links:
    UserRepositories:
      # returns array of '#/components/schemas/repository'
      operationId: getRepositoriesByOwner
      parameters:
        username: $response.body#/username
    UserRepository:
      # returns '#/components/schemas/repository'
      operationId: getRepository
      parameters:
        username: $response.body#/owner/username
        slug: $response.body#/slug
    RepositoryPullRequests:
      # returns '#/components/schemas/pullrequest'
      operationId: getPullRequestsByRepository
      parameters:
        username: $response.body#/owner/username
        slug: $response.body#/slug
    PullRequestMerge:
      # executes /2.0/repositories/{username}/{slug}/pullrequests/{pid}/merge
      operationId: mergePullRequest
      parameters:
        username: $response.body#/author/username
        slug: $response.body#/repository/slug
        pid: $response.body#/id
  schemas:
    user:
      type: object
      properties:

```

```

    username:
      type: string
    uuid:
      type: string
  repository:
    type: object
    properties:
      slug:
        type: string
      owner:
        $ref: '#/components/schemas/user'
  pullrequest:
    type: object
    properties:
      id:
        type: integer
      title:
        type: string
      repository:
        $ref: '#/components/schemas/repository'
      author:
        $ref: '#/components/schemas/user'

```

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

```

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

```

or an absolute operationRef:

```

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

```

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

21. 標頭物件 (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 with of an integer type:

```
{
  "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
```

22. 標籤物件 (Tag Object)

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

Fixed Fields

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

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

Tag Object Example

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

23. (複)範例物件 (Examples Object)

Anywhere an example may be given, a JSON Reference MAY be used, with the explicit restriction that examples having a JSON format with object named \$ref are not allowed. This does mean that example, structurally, can be either a string primitive or an object, similar to additionalProperties.

In all cases, the payload is expected to be compatible with the type schema for the value that it is accompanying. Tooling implementations MAY choose to validate compatibility automatically, and reject the example value(s) if they are not compatible.

```
# in a model
schemas:
  properties:
    name:
      type: string
      example:
        $ref: http://foo.bar#/examples/name-example

# in a request body, note the plural `examples` as the Content-Type is set to `*`:
requestBody:
  content:
    'application/json':
      schema:
        $ref: '#/components/schemas/Address'
      examples:
        - {"foo": "bar"}
        - {"bar": "baz"}
    'application/xml':
      examples:
        - $ref: 'http://foo.bar#/examples/address-example.xml'
```

```

    'text/plain':
      examples:
        - $ref: 'http://foo.bar#/examples/address-example.txt'

# in a parameter
parameters:
  - name: 'zipCode'
    in: 'query'
    schema:
      type: 'string'
      format: 'zip-code'
      example:
        $ref: 'http://foo.bar#/examples/zip-example'

# in a response, note the plural `examples`:
responses:
  '200':
    description: your car appointment has been booked
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/SuccessResponse'
        example:
          $ref: http://foo.bar#/examples/address-example.json

```

24. 参考物件 (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 done 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 File Example

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

Relative Files With Embedded Schema Example

```
{
  "$ref": "definitions.json#/Pet"
}
$ref: definitions.yaml#/Pet
```

25. Schema 物件 (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](#).

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

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.
- `description` - [CommonMark syntax](#) can 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 in place 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
<code>nullable</code>	<code>boolean</code>	Allows sending a <code>null</code> value for the defined schema. Default value is <code>false</code> .
<code>discriminator</code>	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.
<code>readOnly</code>	<code>boolean</code>	Relevant only for Schema <code>"properties"</code> definitions. Declares the property as <code>"read only"</code> . This means that it MAY be sent as part of a response but SHOULD NOT be sent as part of the request. If property is marked as <code>readOnly</code> being <code>true</code> and is in the required list, the required will take effect on the response only. A property MUST NOT be marked as both <code>readOnly</code> and <code>writeOnly</code> being <code>true</code> . Default value is <code>false</code> .

Field Name	Type	Description
writeOnly	boolean	Relevant only for Schema "properties" definitions. Declares the property as "write only". This means that it MAY be sent as part of a request but SHOULD NOT be sent as part of the response. If 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 format of this property.
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 naturally be 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 can 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 in an array of object definitions that are validated independently but together compose a single object.

While composition offers model extensibility, it does not imply a hierarchy between the models. To support polymorphism, OpenAPI Specification adds the support of the `discriminator` field. When used, the `discriminator` will be the name of the property used to decide which schema definition is used to validate the structure of the model. As such, the `discriminatorField` MUST be a required field. There are two ways to define the value of a discriminator for an inheriting instance.

- Use the schema's name.
- Override the schema's name by overriding the property with a new value. If exists, this takes precedence over the schema's name. As such, inline schema definitions, which do not have a given id, cannot be used in polymorphism.

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

```

26. 識別物件 (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.

Note, when using the discriminator, inline schemas will not be considered when using the discriminator.

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 attribute 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: pet_type

The expectation now is that a property with name `pet_type` 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,
  "pet_type": "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: pet_type

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 a implicit or explicit mapping, no schema can be determined and validation SHOULD fail. Note, mapping keys MUST be string values, but tooling MAY 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 composing the parent schema in an allOfconstruct may be used as an alternate schema.

For example:

components:

schemas:

Pet:

type: object

required:

- pet_type

properties:

pet_type:

type: string

discriminator:

```

    propertyName: pet_type
    mapping:
      cachorro: 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:

```

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

```

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

```

{
  "pet_type": "cachorro",
  "bark": "soft"
}

```

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

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

This object can 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>

```

Changing the element names:

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

The external name property has no effect on the XML:

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

```

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

```

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

```

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

```

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

```

{
  "animals": {
    "type": "array",
    "items": {

```

```

    "type": "string",
    "xml": {
      "name": "animal"
    }
  },
  "xml": {
    "wrapped": true
  }
}

```

```

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>

```

28. 安全方案物件 (Security Scheme Object)

Allows the definition of a security scheme that can be used by the operations.

Supported schemes are HTTP authentication, an API key (either as a header or as a query parameter) and OAuth2's common flows (implicit, password, application and access code).

Fixed Fields

Field Name	Type	Validity	Description
type	string	Any	REQUIRED. The type of the security scheme. Valid values are "apiKey", "http", "oauth2", "openIdConnect".
description	string	Any	A short description for security scheme. CommonMark syntax can be used for rich text representation.
name	string	apiKey	REQUIRED. The name of the header or query parameter to be used.
in	string	apiKey	REQUIRED. The location of the API key. Valid values are "query" or "header".
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 can 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
```

29. OAuth 流程物件 (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 can be extended with [Specification Extensions](#).

30. OAuth 單一流程物件 (OAuth Flow Object)

Configuration details for a supported OAuth Flow

Fixed Fields

Field Name	Type	Validity	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.
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 scope name and a short description for

Field Name	Type	Validity	Description
			it.

This object can 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`

31. 安全需求物件 (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 [Open API object](#) or [Operation Object](#), only one of Security Requirement Objects in the list needs to be satisfied to authorize.

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

七、規格擴充 (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).

八、安全篩選 (Security Filtering)

在 OAS 標準中，某些物件可能被宣告(或維持)為空值，或是完全移除，即使這些物件是 API 文件的核心要素。

理由是為了允許對文件本身設定另一層擷取控制。雖然並非標準的一部分，但某些函式庫可能會選擇允許基於某種形式的身份驗證/授權來擷取文件的部分內容。

以下提供兩個範例：

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

附錄 A: 歷史版本

Version	Date	Notes
3.0.0-rc1	2017-04-27	rc1 of the 3.0 specification
3.0.0-rc0	2017-02-28	Implementer's Draft of the 3.0 specification
2.0	2015-12-31	Donation of Swagger 2.0 to the Open API Initiative
2.0	2014-09-08	Release of Swagger 2.0
1.2	2014-03-14	Initial release of the formal document.
1.1	2012-08-22	Release of Swagger 1.1
1.0	2011-08-10	First release of the Swagger Specification

原始文件來源：

<https://github.com/OAI/OpenAPI-Specification/blob/OpenAPI.next/versions/3.0.md>