OASIS Committee Note

OData to OpenAPI Mapping Version 1.0

Committee Note 01

11 July 2019

This stage:
https://docs.oasis-open.org/odata/odata-openapi/v1.0/cn01/odata-openapi-v1.0-cn01.docx (Authoritative)
https://docs.oasis-open.org/odata/odata-openapi/v1.0/cn01/odata-openapi-v1.0-cn01.html
https://docs.oasis-open.org/odata/odata-openapi/v1.0/cn01/odata-openapi-v1.0-cn01.pdf

Previous stage:
http://docs.oasis-open.org/odata/odata-openapi/v1.0/cnprd01/odata-openapi-v1.0-cnprd01.docx (Authoritative)
http://docs.oasis-open.org/odata/odata-openapi/v1.0/cnprd01/odata-openapi-v1.0-cnprd01.html
http://docs.oasis-open.org/odata/odata-openapi/v1.0/cnprd01/odata-openapi-v1.0-cnprd01.pdf

Latest stage:
https://docs.oasis-open.org/odata/odata-openapi/v1.0/odata-openapi-v1.0.docx (Authoritative)
https://docs.oasis-open.org/odata/odata-openapi/v1.0/odata-openapi-v1.0.html
https://docs.oasis-open.org/odata/odata-openapi/v1.0/odata-openapi-v1.0.pdf

Technical Committee:
OASIS Open Data Protocol (OData) TC

Chairs:
Ralf Handl (ralf.handl@sap.com), SAP SE
Mike Pizzo (mikep@microsoft.com), Microsoft

Editors:
Ralf Handl (ralf.handl@sap.com), SAP SE
Hubert Heijkers (hubert.heijkers@nl.ibm.com), IBM
Mike Pizzo (mikep@microsoft.com), Microsoft
Martin Zurmuehl (martin.zurmuehl@sap.com), SAP SE

Related work:
This document is related to:

- *OData Version 4.01. Part 1: Protocol*. Latest version. http://docs.oasis-open.org/odata/odata/v4.01/odata-v4.01-part1-protocol.html.
- *OData Version 4.01. Part 2: URL Conventions*. Latest version. http://docs.oasis-open.org/odata/odata/v4.01/odata-v4.01-part2-url-conventions.html.
- *OData Common Schema Definition Language (CSDL) XML Representation Version 4.01*. Latest Version. http://docs.oasis-open.org/odata/odata-csdl-xml/v4.01/odata-csdl-xml-v4.01.html
- *OData JSON Format Version 4.01*. Latest Version. http://docs.oasis-open.org/odata/odata-json-format/v4.01/odata-json-format-v4.01.html
- *OData Vocabularies Version 4.0*. Latest Version. http://docs.oasis-open.org/odata/odata-vocabularies/v4.0/odata-vocabularies-v4.0.html

Abstract:

The Open Data Protocol (OData) is an open protocol for creating and consuming queryable and interoperable RESTful APIs in a simple and standard way. OData services are described by an entity-relationship model, and the model description is an integral part of each OData service.

The OpenAPI Specification (OAS) is a standard, language-agnostic interface to REST APIs which allows both humans and computers to discover and understand the capabilities of the service. This document describes a possible mapping of OData service descriptions to OAS documents.

Status:

Citation format:

When referencing this document the following citation format should be used:

[OData-OpenAPI-v1.0]

OData to OpenAPI Mapping Version 1.0. Edited by Ralf Handl, Hubert Heijkers, Mike Pizzo, and Martin Zurmuehl. 11 July 2019. OASIS Committee Note 01. https://docs.oasis-open.org/odata/odata-openapi/v1.0/cn01/odata-openapi-v1.0-cn01.html. Latest version: https://docs.oasis-open.org/odata/odata-openapi/v1.0/odata-openapi-v1.0.html.

Notices

Table of Contents

1 Introduction

The Open Data Protocol (OData) is an application-level protocol for interacting with data via RESTful APIs. It supports the description of data models and the editing and querying of data according to those models. The OData Protocol is different from other REST-based web service approaches in that it provides a uniform way to describe both the data and the data model. This improves semantic interoperability between systems. The Entity Data Model (EDM) describing an OData service is part of the service itself. **[OData-CSDL]** defines an XML representation of the entity data model exposed by an OData service.

The OpenAPI Specification (OAS, formerly known as Swagger RESTful API Documentation Specification) defines a standard, language-agnostic interface to RESTful APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. When properly defined, a consumer can understand and interact with the remote service with a minimal amount of implementation logic. An OpenAPI definition can then be used by documentation generation tools to display the API, code generation tools to generate servers and clients in various programming languages, testing tools, and many other use cases.

This document describes a possible mapping of OData service descriptions to OAS documents which allows OpenAPI tools to be used for interacting with OData services.

OData is based on a powerful set of concepts and conventions which allow rich interaction with OData services. OpenAPI on the other hand does not assume or rely on any conventions and requires explicit and – from an OData perspective – relatively low-level and repetitive description of each service feature. As a consequence, this mapping only translates the basic features of an OData service into OpenAPI terms to allow an easy "first contact" by exploring it e.g. with the Swagger UI **[Swagger UI]**, rather than trying to capture all features of an OData service in an unmanageably long OAS document.

1.1 References (non-normative)

[OData-CSDL]	*OData Common Schema Definition Language (CSDL) XML Representation Version 4.01*. See link in "Related work" section on cover page.
[OData-JSON]	*OData JSON Format Version 4.01.* See link in "Related work" section on cover page.
[OData-OpenAPI]	*odata-openapi OASIS TC GitHub repository* https://github.com/oasis-tcs/odata-openapi.
[OData-Protocol]	*OData Version 4.01 Part 1: Protocol*. See link in "Related work" section on cover page.
[OData-URL]	*OData Version 4.01 Part 2: URL Conventions*. See link in "Related work" section on cover page.
[OData-VocAuth]	*OData Vocabularies Version 4.0: Authorization Vocabulary.* See link in "Related work" section on cover page.
[OData-VocCap]	*OData Vocabularies Version 4.0: Capabilities Vocabulary.* See link in "Related work" section on cover page.
[OData-VocCore]	*OData Vocabularies Version 4.0: Core Vocabulary.* See link in "Related work" section on cover page.
[OData-VocVal]	*OData Vocabularies Version 4.0: Core Vocabulary.* See link in "Related work" section on cover page.
[OpenAPI]	*OpenAPI Specification Version 3.0.2*, https://openapis.org/specification, specifically https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.1.md
[RFC8259]	Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 8259, December 2017. http://tools.ietf.org/html/rfc827159.
[Swagger UI]	https://github.com/swagger-api/swagger-ui

[YAML] YAML Ain't Markup Language (YAML^{TM}), Version 1.2, 3rd Edition, Patched at 2001-10-01. Copyright © 2001-2009 Oren Ben-Kiki, Clark Evans, Ingy döt Net. http://www.yaml.org/spec/1.2/spec.html

2 Design Principles

Given the different goals of and levels of abstractions used by OData and OpenAPI, this mapping of OData metadata documents into OAS documents is intentionally lossy and only tries to preserve the main features of an OData service:

- The entity container is translated into an OpenAPI Paths Object with path templates and operation objects for all top-level resources described by the entity container
- Structure-describing CSDL elements (structured types, type definitions, enumerations) are translated into OpenAPI Schema Objects within the OpenAPI Components Object
- CSDL constructs that don't have an OpenAPI counterpart are omitted

3 Providing OAS Documents for an OData Service

OAS documents describing an OData `service` can be provided in several ways, and the examples given here are by no means exhaustive or mutually exclusive.

Typical provisioning is as a static resource, e.g. as part of a Service Catalog or API Hub.

A more OData-ish way is to provide the OAS document as part of the service. Following the OpenAPI convention, this would be a resource `<service-root>/openapi.json` at the service root, next to `<service-root>/$metadata`.

4 OAS Document Structure

OAS documents are represented as JSON objects and conform to **[RFC8259]**. **[YAML]**, being a superset of JSON, can be used as well to represent an OAS document.

An OAS document consists of a single OpenAPI Object, see **[OpenAPI]**. It is represented as a JSON object. How to construct each of its name/value pairs ("fields" in OpenAPI terminology) is described in the following sections.

EXAMPLE 1: STRUCTURE OF AN OAS DOCUMENT

```
{
  "openapi":"3.0.1",
  "info": …,
  "servers": …,
  "tags": …,
  "paths": …,
  "components": …
}
```

4.1 Field `openapi`

The value of `openapi` is a string representing the OpenAPI version used, e.g. `"3.0.2"`.

4.2 Field `info`

The value of `info` is an Info Object, see **[OpenAPI]**. It contains the fields `title` and `version`, and optionally the field `description`.

EXAMPLE 2: INFO OBJECT – NOTE THAT `DESCRIPTION` ALLOWS COMMONMARK SYNTAX

```
"info":{
  "title":"OData Service for namespace ODataDemo",
  "version":"0.1.0",
  "description":"This OData service is located at [http://localhost/service-
root/](http://localhost/service-root/)\n\n## References\n-
[Org.OData.Core.V1](https://github.com/oasis-tcs/odata-
vocabularies/blob/master/vocabularies/Org.OData.Core.V1.md)\n-
[Org.OData.Measures.V1](https://github.com/oasis-tcs/odata-
vocabularies/blob/master/vocabularies/Org.OData.Measures.V1.md)"
}
```

4.2.1 Field `title`

The value of `title` is the value of the unqualified annotation `Core.Description` (see **[OData-VocCore]**) of the main schema or the entity container of the OData service.

If no `Core.Description` is present, a default title must be provided as this is a required OpenAPI field.

4.2.2 Field `version`

The value of `version` is the value of the annotation `Core.SchemaVersion` (see **[OData-VocCore]**) of the main schema.

If no `Core.SchemaVersion` is present, a default version must be provided as this is a required OpenAPI field.

4.2.3 Field `description`

The value of `description` is the value of the annotation `Core.LongDescription` (see **[OData-VocCore]**) of the main schema or the entity container.

While this field is optional, it prominently appears in OpenAPI exploration tools, so a default description should be provided if no `Core.LongDescription` annotation is present.

4.3 Field `servers`

The value of `servers` is an array of Server Objects, see **[OpenAPI]**. It contains one object with a field `url`. The value of `url` is a string containing the service root URL without the trailing forward slash.

EXAMPLE 3: ABSOLUTE SERVICE ROOT URL

```
"servers": [
  {
    "url": "http://localhost/service-root"
  }
]
```

The value of `url` can be relative to the discovery document `openapi.json`, so if it is located at the service root, the relative URL is just a dot.

EXAMPLE 4: RELATIVE SERVICE ROOT URL IF `OPENAPI.JSON` IS LOCATED AT THE SERVICE ROOT

```
"servers": [
  {
    "url": "."
  }
]
```

4.4 Field `tags`

The value of `tags` is an array of Tag Objects, see **[OpenAPI]**. Tags are used for logical grouping of operations. For an OData service the natural groups are entity sets and singletons, so the `tags` array contains one Tag Object per entity set and singleton in the entity container.

A Tag Object has to contain the field `name`, whose value is the name of the entity set or singleton, and it optionally can contain the field `description`, whose value is the value of the unqualified annotation `Core.Description` of the entity set or singleton.

The `tags` array can contain additional Tag Objects for other logical groups, e.g. for action imports or function imports that are not associated with an entity set, or for entity types that are only used in containment navigation properties or action/function return types.

EXAMPLE 5: TAGS WITH OPTIONAL DESCRIPTIONS

```
"tags": [
  {
    "name": "Products"
  },
  {
    "name": "Categories",
    "description": "Product Categories"
  },
  {
    "name": "Suppliers"
  },
  {
    "name": "MainSupplier",
    "description": "Primary Supplier"
  },
  {
    "name": "Countries"
  }
]
```

4.5 Field `paths`

The value of `paths` is a Paths Object, see **[OpenAPI]**. It is the main source of information on how to use the described API. It consists of name/value pairs whose name is a path template relative to the service root URL, and whose value is a Path Item Object, see **[OpenAPI]**.

Due to the power and flexibility of OData a full representation of all service capabilities in the Paths Object is typically not feasible, so this mapping only describes the minimum information desired in the Paths Object. Implementations are allowed – and in fact encouraged – to add additional information that is deemed useful for the intended target audience of the OpenAPI description of that service, leveraging the documentation features of the OpenAPI Specification, especially and not limited to human-readable descriptions.

The minimum information to be included in the Paths Object is described in the remainder of this section. The Paths Object reflects the top-level resources and capabilities of the service as closely as possible, i.e. only list supported operations and query options.

EXAMPLE 6: PATHS FOR ENTITY SETS, INDIVIDUAL ENTITIES, SINGLETONS, ACTION IMPORTS, AND FUNCTION IMPORTS USING PARENS-STYLE KEY SYNTAX

```
"paths": {
  "/Products": …,
  "/Products('{ID}')": …,
  "/Categories": …,
  "/Categories({ID})": …,
  "/Suppliers": …,
  "/Suppliers('{ID}')": …,
  "/MainSupplier": …,
  "/Countries": …,
  "/Countries('{Code}')": …,
  "/ProductsByRating(Rating={Rating})": …
}
```

OData 4.01 introduces semantics for representing a key as a segment in a URL path, see term `Capabilities.KeyAsSegmentSupported` in **[OData-VocCap]**. While compliant services are also required to support the traditional parentheses-style syntax shown above for interoperability, services that additionally support the key-as-segment syntax may choose to advertise either or both styles of supported paths in OpenAPI. The examples used in the remainder of this document favor the key-as-segment style convention, as shown in the example below.

EXAMPLE 7: PATHS USING KEY-AS-SEGMENT CONVENTION FOR ENTITIES AND IMPLICIT PARAMETER ALIASES FOR FUNCTION IMPORTS

```
"paths": {
  "/Products": …,
  "/Products/{ID}": …,
  "/Categories": …,
  "/Categories/{ID}": …,
  "/Suppliers": …,
  "/Suppliers/{ID}": …,
  "/MainSupplier": …,
  "/Countries": …,
  "/Countries/{Code}": …,
  "/ProductsByRating": …
}
```

4.5.1 Paths for Collections of Entities

Each entity set is represented as a name/value pair whose name is the service-relative resource path of the entity set prepended with a forward slash, and whose value is a Path Item Object, see **[OpenAPI].**

EXAMPLE 8: PATH TEMPLATE FOR AN ENTITY SET

```
"/Products": …
```

Each collection-valued navigation property of the declared entity type of an entity set, one of its derived types, or of a complex property (recursively) of one of these entity types, is represented as a name/value pair whose name is the path template for reaching that navigation property, with path parameters for the key values. The Path Item Object contains a `parameters` array with one Parameter Object for each key property in the path template. These Parameter Objects use the same type mapping as described for primitive properties. They optionally can contain the field `description` whose value is the value of the unqualified annotation `Core.Description` of the key property.

EXAMPLE 9: PATH TEMPLATE FOR A NAVIGATION PROPERTY

```
"/Categories/{ID}/Products": …
```

Each bound action or function applicable to the collection is represented as an additional name/value pair with a path template for the action/function invocation, with path parameters for the key values in the binding path, and with path parameters for non-binding function parameters. If the action or function is defined in a namespace that is marked as a default namespace through the `Core.DefaultNamespace` term (see **[OData-VocCore]**), path templates both with and without namespace prefix may be provided. Care has to be taken to avoid ambiguous path templates as **[OpenAPI]** doesn't define how to resolve ambiguities and leaves this to the individual tool authors.

EXAMPLE 10: PATH TEMPLATE FOR A FUNCTION BOUND TO A COLLECTION OF ENTITIES

```
"/Products/OData.Demo.BestSelling()": …
```

EXAMPLE 11: PATH TEMPLATE FOR A BOUND FUNCTION WITHOUT NAMESPACE AND PARENTHESES

```
"/Products/BestSelling": …
```

EXAMPLE 12: PATH TEMPLATE WITH KEY VALUES IN BINDING PATH AND BOUND FUNCTION

```
"/Categories/{ID}/Products/OData.Demo.BestSelling()": …
```

4.5.1.1 Query a Collection of Entities

The Path Item Object for a collection of entities that supports querying (see terms `Capabilities.ReadRestrictions` and `Capabilities.NavigationRestrictions` in **[OData-VocCap]**) contains the keyword `get` with an Operation Object as value that describes the capabilities for querying the collection. The `tags` array of the Operation Object includes the entity set name in the first segment of the path template. Additional tag values, e.g. for the entity type of a containment navigation property or the target entity set of a non-containment navigation property, can be included to make this operation more easily discoverable.

Example 13: GET operation for an entity set – summary and tags

```
"/Products": {
  "get": {
    "summary": "Get entities from Products",
    "tags": [
      "Products"
    ],
```

The `parameters` array contains Parameter Objects for all system query options allowed for this collection. It does not list system query options not allowed for this collection, see **[OData-VocCap]** terms

- `Capabilities.CountRestrictions`
- `Capabilities.ExpandRestrictions`
- `Capabilities.FilterRestrictions`
- `Capabilities.NavigationRestrictions/RestrictedProperties`
- `Capabilities.SearchRestrictions`
- `Capabilities.SelectSupport`
- `Capabilities.SkipSupported`
- `Capabilities.SortRestrictions`
- `Capabilities.TopSupported`

Example 14: GET operation for an entity set - parameters

```
    "parameters": [
      {
        "$ref": "#/components/parameters/top"
      },
      {
        "$ref": "#/components/parameters/skip"
      },
      {
        "$ref": "#/components/parameters/search"
      },
      {
        "$ref": "#/components/parameters/filter"
      },
      {
        "$ref": "#/components/parameters/count"
      },
```

Note: the syntax of the system query options `$expand`, `$select`, and `$orderby` is too flexible to be formally described with OpenAPI Specification means, yet the typical use cases of just providing a comma-separated list of properties can be expressed via an array-valued parameter with an `enum` constraint, as shown in the following example. This makes it easy to try out these system query options in OpenAPI tools.

Example 15: GET operation for an entity set – more specific parameters

```
      {
        "name": "$expand",
        "in": "query",
        "description": "Expand related entities, see [OData
Expand](http://docs.oasis-open.org/odata/odata/v4.01/cs01/part1-
protocol/odata-v4.01-cs01-part1-protocol.html#sec_SystemQueryOptionexpand)",
        "explode": false,
        "schema": {
          "type": "array",
          "uniqueItems": true,
          "items": {
            "type": "string",
            "enum": [
              "*",
              "Category",
              "Supplier"
            ]
          }
        }
      },
      {
        "name": "$select",
        "in": "query",
        "description": "Select properties to be returned, see [OData
Select](http://docs.oasis-open.org/odata/odata/v4.01/cs01/part1-
protocol/odata-v4.01-cs01-part1-protocol.html#sec_SystemQueryOptionselect)",
        "explode": false,
        "schema": {
          "type": "array",
          "uniqueItems": true,
          "items": {
            "type": "string",
            "enum": [
              "*",
              "ID",
              "Description",
              "ReleaseDate",
              "DiscontinuedDate",
              "Rating",
              "Price",
              "Currency"
            ]
          }
        }
      },
      {
        "name": "$orderby",
        "in": "query",
        "description": "Order items by property values, see [OData
Sorting](http://docs.oasis-open.org/odata/odata/v4.01/cs01/part1-
protocol/odata-v4.01-cs01-part1-protocol.html#sec_SystemQueryOptionorderby)",
        "explode": false,
        "schema": {
          "type": "array",
          "uniqueItems": true,
          "items": {
            "type": "string",
            "enum": [
              "ID",
              "ID desc",
              "Description",
              "Description desc",
              "ReleaseDate",
              "ReleaseDate desc",
              "DiscontinuedDate",
              "DiscontinuedDate desc",
              "Rating",
              "Rating desc",
              "Price",
              "Price desc",
              "Currency",
              "Currency desc"
            ]
          }
        }
      }
    ],
```

The value of `responses` is a Responses Object, see **[OpenAPI]**. It contains a name/value pair for the success case (HTTP response code `200`) describing the structure of a successful response referencing the schema of the collection's entity type in the global `schemas`.

In addition, it contains a `default` name/value pair for the OData error response referencing the global `responses`.

Example 16: GET operation for an entity set - responses

```
    "responses": {
      "200": {
        "description": "Retrieved entities",
        "content": {
          "application/json": {
            "schema": {
              "type": "object",
              "title": "Collection of Product",
              "properties": {
                "value": {
                  "type": "array",
                  "items": {
                    "$ref": "#/components/schemas/ODataDemo.Product"
                  }
                }
              }
            }
          }
        }
      },
      "default": {
        "$ref": "#/components/responses/error"
      }
    }
  }
},
```

4.5.1.2 Create an Entity

If the collection of entities allows inserts (see terms `Capabilities.InsertRestrictions` and `Capabilities.NavigationRestrictions` in **[OData-VocCap]**), the Path Item Object contains the keyword `post` with an Operation Object as value that describes the capabilities for creating new entities. The `tags` array of the Operation Object includes the entity set name in the first segment of the path template. Additional tag values, e.g. for the entity type of a containment navigation property or the target entity set of a non-containment navigation property, can be included to make this operation more easily discoverable.

The `parameters` array contains a Parameter Object for each key property in the path template, and it contains specific Parameter Objects for the system query options `$select` and `$expand` if these are allowed.

The `requestBody` field contains a Request Body Object for the request body that references the schema of the entity set's entity type in the global `schemas`.

The `responses` object contains a name/value pair for the success case (HTTP response code `201`) describing the structure of the success response referencing the schema of the entity type in the global `schemas`. If the service supports the preference `return=minimal`, it contains a name/value pair for the HTTP response code `204`. In addition, it contains a `default` name/value pair for the OData error response referencing the global `responses`.

EXAMPLE 17: `POST` OPERATION FOR AN ENTITY SET

```
"post": {
  "summary": "Add new entity to Products",
  "tags": [
    "Products"
  ],
  "requestBody": {
    "required": true,
    "description": "New entity",
    "content": {
      "application/json": {
        "schema": {
          "$ref": "#/components/schemas/ODataDemo.Product"
          }
        }
      }
    },
    "responses": {
      "201": {
        "description": "Created entity",
        "content": {
          "application/json": {
            "schema": {
              "$ref": "#/components/schemas/ODataDemo.Product"
            }
          }
        }
      },
      "204": {
        "description": "Success"
      },
      "default": {
        "$ref": "#/components/responses/error"
        }
      }
    }
  }
}
```

4.5.1.3 Invoke Bound Actions and Functions on Collections

The Path Item Object for a bound action contains the keyword `post`, the Path Item Object for a bound function contains the keyword `get`. The value of the `post` or `get` keyword is an Operation Object that describes how to invoke the action or function. The `tags` array of the Operation Object includes the entity set name in the first segment of the path template. Additional tag values, e.g. for the binding parameter type or the result type of the action or function, can be included to make this operation more easily discoverable.

EXAMPLE 18: FUNCTION BOUND TO A COLLECTION OF ENTITIES – `SUMMARY` AND `TAGS`

```
"/Categories/{ID}/Products/OData.Demo.BestSelling()": {
  "get": {
    "summary": "Determine the best-selling product",
    "tags": [
      "Categories",
      "Products"
    ],
```

The `parameters` array contains a Parameter Object for each key property and for each non-binding parameter in the path template.

The Parameter Objects describing key values and primitive parameter values use the same type mapping as described for primitive properties.

Structured or collection-valued parameters are represented as a parameter alias in the path template and the `parameters` array contains a Parameter Object for the parameter alias as a query option of type `string`. The parameter `description` describes the format of this URL-encoded JSON object or array, and/or references to **[OData-URL]**, section "Complex and Collection Literals".

Parameter Objects optionally can contain the field `description`, whose value is the value of the unqualified annotation `Core.Description` of the parameter.

Depending on the result type of the bound action or function the `parameters` array contains specific Parameter Objects for the allowed system query options.

For bound actions on collections that use optimistic concurrency control and require ETags for modification operations (see term `Core.OptimisticConcurrency` in **[OData-VocCore]**), the `parameters` array contains a Parameter Object for the `If-Match` header.

For bound actions with non-binding parameters, the `requestBody` field contains a Request Body Object describing the structure of the request body. Its `schema` value follows the rules for Schema Objects for complex types, with one property per non-binding action parameter.

EXAMPLE 19: FUNCTION BOUND TO A COLLECTION OF ENTITIES – `PARAMETERS`

```
"parameters": [
  {
    "name": "ID",
    "in": "path",
    "required": true,
    "description": "key: ID",
    "schema": {
      "type": "integer",
      "format": "int32"
    }
  },
  {
    "name": "$expand",
    "in": "query",
    …
  },
  {
    "name": "$select",
    "in": "query",
    …
  }
],
```

The `responses` object contains name/value pairs for the success cases (HTTP response code `204` if the response has no body or the service supports the preference `return=minimal`, `201` in case a new entity is created by the operation, and 200 in other success cases). In addition, it contains a `default` name/value pair for the OData error response referencing the global `responses`.

EXAMPLE 20: ACTION BOUND TO ENTITY WITHIN A SET – `RESPONSES`

```
"responses": {
  "200": {
    "description": "The best-selling product",
    "content": {
          "application/json": {
            "schema": {
              "$ref": "#/components/schemas/ODataDemo.Product"
            }
          }
        }
      },
      "default": {
        "$ref": "#/components/responses/error"
      }
    }
  }
}
```

4.5.2 Paths for Single Entities

Each entity set that is indexable by key (see term `Capabilities.IndexableByKey` and `Capabilities.NavigationRestrictions` in **[OData-VocCap]**) is represented as a name/value pair whose name is the path template for key access, with path parameters for the key values, and whose value is a Path Item Object describing the allowed operations on individual entities of this set. It contains a `parameters` array with one Parameter Object for each key property in the path template. These Parameter Objects use the same type mapping as described for primitive properties. They optionally can contain the field `description` whose value is the value of the unqualified annotation `Core.Description` of the key property.

If the service supports both parentheses-style key predicates and key-as-segment convention (indicated by annotating the entity container with term `Capabilities.KeyAsSegmentSupported` in **[OData-VocCap]**), the service author is free to provide path templates for both key conventions or just for the convention preferred by the service author.

EXAMPLE 21: PATH TEMPLATE FOR AN INDIVIDUAL ENTITY WITHIN AN ENTITY SET – SINGLE-PART KEY

```
"/Products('{ID}')": …
```

EXAMPLE 22: PATH TEMPLATE FOR AN INDIVIDUAL ENTITY USING KEY-AS-SEGMENT CONVENTION

```
"/Products/{ID}": …
```

EXAMPLE 23: PATH TEMPLATE FOR AN INDIVIDUAL ENTITY WITHIN AN ENTITY SET – MULTI-PART KEY

```
"/OrderItems(OrderID={OrderID},ItemID={ItemID})": …
```

EXAMPLE 24: PATH TEMPLATE FOR MULTI-PART KEY USING KEY-AS-SEGMENT CONVENTION

```
"/OrderItems/{OrderID}/{ItemID})": …
```

Each singleton is represented as a name/value pair whose name is the service-relative resource path of the singleton prepended with a forward slash, and whose value is Path Item Object describing the allowed operations on this singleton.

EXAMPLE 25: PATH TEMPLATE FOR A SINGLETON

```
"/MainSupplier": …
```

Each single-valued containment navigation property is represented as a name/value pair whose name is the shortest possible path template for reaching that navigation property, with path parameters for the key values of all intermediate entity types.

Each collection-valued containment navigation property that allows key access is represented as a name/value pair whose name is the shortest possible path template for reaching that navigation property, plus a key segment, with path parameters for all key values. If the contained entity itself has navigation properties, paths for these navigation properties are also added, applying the rules in this and the preceding section recursively, up to an implementation-specific limit.

Each single-valued navigation property of the declared entity type of an entity set, one of its derived types, or of a complex property (recursively) of one of these entity types, is represented as a name/value pair whose name is the path template for reaching that navigation property, with path parameters for the key values.

Example 26: Path template for a single-valued navigation property on an individual entity

```
"/Products/{ID}/Category": …
```

Each bound action or function applicable to a single entity is represented as an additional name/value pair with a path template for the action/function invocation, with path parameters for key values needed to identify the single entity, and with path parameters for non-binding function parameters. If the action or function is defined in a namespace that is marked as a default namespace through the `Core.DefaultNamespace` term (see **[OData-VocCore]**), path templates both with and without namespace prefix may be provided. Care has to be taken to avoid ambiguous path templates as **[OpenAPI]** doesn't define how to resolve ambiguities and leaves this to the individual tool authors.

Example 27: Path template for an action bound to a single entity within an entity set

```
"/LeaveRequests/{ID}/OData.Demo.Approval": …
```

Example 28: Path template for a bound action without namespace prefix

```
"/LeaveRequests/{ID}/Approval": …
```

4.5.2.1 Retrieve an Entity

If the entity allows read (see terms `Capabilities.ReadRestrictions`, `Capabilities.ReadByKeyRestrictions`, and `Capabilities.NavigationRestrictions` in **[OData-VocCap]**), the Path Item Object for a single entity contains the keyword `get` with an Operation Object as value that describes the capabilities for retrieving a single entity. The `tags` array of the Operation Object includes the entity set or singleton name in the first segment of the path template. Additional tag values, e.g. for the entity type of a containment navigation property or the target entity set of a non-containment navigation property, can be included to make this operation more easily discoverable.

Example 29: `GET` operation for an individual entity

```
"/Products/{ID}": {
  "get": {
    "summary": "Get entity from Products by key",
    "tags": [
      "Products"
    ],
```

The `parameters` array contains specific Parameter Objects for the system query options `$select` and `$expand` if these are allowed.

Example 30: `GET` operation for an individual entity - `parameters`

```
    "parameters": [
      {
        "name": "ID",
        "in": "path",
        "required": true,
        "description": "key: ID",
        "schema": {
          "type": "string"
        }
      },
      {
        "name": "$select",
        "in": "query",
        "description": "Select properties to be returned, see [OData 
Select](http://docs.oasis-
open.org/odata/odata/v4.0/errata02/os/complete/part1-protocol/odata-v4.0-
errata02-os-part1-protocol-complete.html#_Toc406398297)",
        "explode": false,
        "schema": {
          "type": "array",
          "uniqueItems": true,
          "items": {
            "type": "string",
            "enum": [
              "*",
              "ID",
              "Description",
              "ReleaseDate",
              "DiscontinuedDate",
              "Rating",
              "Price",
              "Currency"
            ]
          }
        }
      },
      {
        "name": "$expand",
        "in": "query",
        "description": "Expand related entities, see [OData 
Expand](http://docs.oasis-
open.org/odata/odata/v4.0/errata02/os/complete/part1-protocol/odata-v4.0-
errata02-os-part1-protocol-complete.html#_Toc406398298)",
        "explode": false,
        "schema": {
          "type": "array",
          "uniqueItems": true,
          "items": {
            "type": "string"
            "enum": [
              "*",
              "Category",
              "Supplier"
            ]
          }
        }
      }
    ],
```

The `responses` object contains a name/value pair for the success case (HTTP response code `200`) describing the structure of the success response referencing the schema of the entity's type in the global `schemas`. In addition, it contains a `default` name/value pair for the OData error response referencing the global `responses`.

EXAMPLE 31: GET OPERATION FOR AN INDIVIDUAL ENTITY - RESPONSES

```
    "responses": {
      "200": {
        "description": "Retrieved entity",
        "content": {
          "application/json": {
            "schema": {
              "$ref": "#/components/schemas/ODataDemo.Product"
            }
          }
        }
      },
      "default": {
        "$ref": "#/components/responses/error"
        }
      }
    }
  },
```

4.5.2.2 Update an Entity

If the entity allows updates (see term `Capabilities.UpdateRestrictions` and `Capabilities.NavigationRestrictions` in **[OData-VocCap]**), the Path Item Object contains the keyword `patch` with an Operation Object as value that describes the capabilities for updating the entity. The `tags` array of the Operation Object includes the entity set or singleton name in the first segment of the path template. Additional tag values, e.g. for the entity type of a containment navigation property or the target entity set of a non-containment navigation property, can be included to make this operation more easily discoverable.

The `parameters` array contains specific Parameter Objects for the system query options `$select` and `$expand` if these are allowed.

If the entity uses optimistic concurrency control and requires ETags for modification operations (see term `Core.OptimisticConcurrency` in **[OData-VocCore]**), the `parameters` array contains a Parameter Object for the `If-Match` header.

The `requestBody` field contains a Request Body Object for the request body that references the schema of the entity's type in the global `schemas`.

The `responses` object contains a name/value pair for the success case (HTTP response code `204`). If the service supports the preference `return=representation`, it contains a name/value pair for the HTTP response code `200` describing the structure of the success response referencing the schema of the entity's type in the global `schemas`. In addition, it contains a `default` name/value pair for the OData error response referencing the global `responses`.

EXAMPLE 32: `PATCH` OPERATION FOR AN INDIVIDUAL ENTITY

```
"patch": {
  "summary": "Update entity in Products",
  "tags": [
    "Products"
  ],
  "parameters": [
    {
      "name": "ID",
      "in": "path",
      "required": true,
      "description": "key: ID",
      "schema": {
        "type": "string"
      }
    },
    {
      "name": "If-Match",
      "in": "header",
      "description": "ETag",
      "schema": {
        "type": "string"
      }
    }
  ],
  "requestBody": {
    "required": true,
    "description": "New property values",
    "content": {
      "application/json": {
        "schema": {
          "$ref": "#/components/schemas/ODataDemo.Product"
        }
      }
    }
  },
  "responses": {
    "204": {
      "description": "Success"
    },
    "default": {
      "$ref": "#/components/responses/error"
      }
    }
  }
},
```

4.5.2.3 Delete an Entity

If the entity allows deletion (see term `Capabilities.DeleteRestrictions` and `Capabilities.NavigationRestrictions` in **[OData-VocCap]**), the Path Item Object contains the keyword `delete` with an Operation Object as value that describes the capabilities for deleting the entity. The `tags` array of the Operation Object includes the entity set or singleton name in the first segment of the path template. Additional tag values, e.g. for the entity type of a containment navigation property or the target entity set of a non-containment navigation property, can be included to make this operation more easily discoverable.

If the entity uses optimistic concurrency control and requires ETags for modification operations (see term `Core.OptimisticConcurrency` in **[OData-VocCore]**), the `parameters` array contains a Parameter Object for the `If-Match` header.

The `responses` object contains a name/value pair for the success case (HTTP response code `204`). In addition, it contains a `default` name/value pair for the OData error response referencing the global `responses`.

EXAMPLE 33: `DELETE` OPERATION FOR AN INDIVIDUAL ENTITY

```
"delete": {
  "summary": "Delete entity from Products",
  "tags": [
    "Products"
  ],
  "parameters": [
    {
      "name": "ID",
      "in": "path",
      "required": true,
      "description": "key: ID",
      "schema": {
        "type": "string"
      }
    },
    {
      "name": "If-Match",
      "in": "header",
      "description": "ETag",
      "schema": {
        "type": "string"
      }
    }
  ],
  "responses": {
    "204": {
      "description": "Success"
    },
    "default": {
      "$ref": "#/components/responses/error"
      }
    }
    }
  }
}
```

4.5.2.4 Invoke Bound Actions and Functions on Single Entities

The Path Item Object for a bound action contains the keyword `post`, the Path Item Object for a bound function contains the keyword `get`. The value of the `post` or `get` keyword is an Operation Object that describes how to invoke the action or function. The `tags` array of the Operation Object includes the entity set or singleton name in the first segment of the path template. Additional tag values, e.g. for the binding parameter type or the result type of the action or function, can be included to make this operation more easily discoverable.

EXAMPLE 34: ACTION BOUND TO ENTITY WITHIN A SET – `SUMMARY` AND `TAGS`

```
"/LeaveRequests/{ID}/OData.Demo.Rejection": {
  "post": {
    "summary": "Invoke action Rejection",
    "tags": [
      "LeaveRequests"
    ],
```

The `parameters` array contains a Parameter Object for each key property and for each non-binding parameter in the path template. These Parameter Objects follow the rules described in section Invoke Bound Actions and Functions on Collections.

Depending on the result type of the bound action or function the `parameters` array contains specific Parameter Objects for the allowed system query options.

For bound actions on entities that use optimistic concurrency control, i.e. require ETags for modification operations, the `parameters` array contains a Parameter Object for the `If-Match` header.

For bound actions with non-binding parameters, the `requestBody` field contains a Request Body Object describing the structure of the request body. Its `schema` value follows the rules for Schema Objects for complex types, with one property per non-binding action parameter.

The `responses` object contains a name/value pair for the success case (HTTP response code `201`) describing the structure of the success response referencing the schema of the entity type in the global `schemas`. If the service supports the preference `return=minimal`, it contains a name/value pair for the HTTP response code `204`. In addition, it contains a `default` name/value pair for the OData error response referencing the global `responses`.

The `responses` object contains name/value pairs for the success cases (HTTP response code `204` if the response has no body or the service supports the preference `return=minimal`, `201` in case a new entity is created by the operation, and 200 in other success cases). In addition, it contains a `default` name/value pair for the OData error response referencing the global `responses`.

4.5.3 Paths for Action Imports

Each action import is represented as a name/value pair whose name is the service-relative resource path of the action import prepended with a forward slash, and whose value is a Path Item Object containing the keyword `post` with an Operation Object as value that describes how to invoke the action import.

If the action import specifies the `EntitySet` attribute, the `tags` array of the Operation Object includes the entity set name. Additional tag values, e.g. for the result type of the action, can be included to make this operation more easily discoverable.

Depending on the result type of the action import’s action the `parameters` array contains specific Parameter Objects for the allowed system query options.

The `requestBody` field contains a Request Body Object describing the structure of the request body. Its `schema` value follows the rules for Schema Objects for complex types, with one property per action parameter.

The `responses` object contains name/value pairs for the success cases (HTTP response code `204` if the response has no body or the service supports the preference `return=minimal`, `201` in case a new entity is created by the operation, and 200 in other success cases). In addition, it contains a `default` name/value pair for the OData error response referencing the global `responses`.

EXAMPLE 35: ACTION IMPORT

```
"/IncreaseSalaries": {
  "post": {
    "summary": "Invoke action IncreaseSalaries",
    "tags": [
      "Service Operations"
    ],
    "requestBody": {
      "description": "Action parameters",
      "content": {
        "application/json": {
          "schema": {
            "type": "object",
            "properties": {
              "percentage": {
                "anyOf": [
                  { "type": "number" },
                  { "type": "string" }
                ],
                "format": "decimal"
              }
            }
          }
        }
      }
    },
    "responses": {
      "204": {
        "description": "Success"
      },
      "default": {
        "$ref": "#/components/responses/error"
      }
    }
  }
}
```

4.5.4 Paths for Function Imports

Each function import is represented as one name/value pair per unbound function overload whose name is the service-relative resource path template of the function overload, and whose value is a Path Item Object containing the keyword `get` with an Operation Object as value that describes how to invoke the function overload.

If the function import specifies the `EntitySet` attribute, the `tags` array of the Operation Object includes the entity set name. Additional tag values, e.g. for the result type of the function, can be included to make this operation more easily discoverable.

The `parameters` array contains a Parameter Object for each parameter of the function overload. These Parameter Objects follow the rules described in section Invoke Bound Actions and Functions on Collections.

Depending on the result type of the function overload the `parameters` array contains specific Parameter Objects for the allowed system query options.

The `responses` object contains a name/value pair for the success case (HTTP response code `200`) describing the structure of the success response by referencing an appropriate schema in the global `schemas`. In addition, it contains a `default` name/value pair for the OData error response referencing the global `responses`.

EXAMPLE 36: FUNCTION IMPORT

```
"/ProductsByRating(Rating={Rating})": {
  "get": {
    "summary": "Invoke function ProductsByRating",
    "tags": [
      "Products"
    ],
    "parameters": [
      {
        "name": "Rating",
        "in": "path",
        "required": true,
        "schema": {
          "type": "integer",
          "format": "int32"
        }
      }
    ],
    "responses": {
      "200": {
        "description": "Success",
        "content": {
          "application/json": {
            "schema": {
              "title": "Result",
              "type": "object",
              "properties": {
                "value": {
                  "type": "array",
                  "items": {
                    "$ref": "#/components/schemas/ODataDemo.Product"
                  }
                }
              }
            }
          }
        }
      },
      "default": {
        "$ref": "#/components/responses/error"
        }
      }
    }
  }
}
```

4.6 Field `components`

The value of `components` is a Components Object, see **[OpenAPI]**. It holds maps of reusable `schemas` describing message bodies, operation `parameters`, and `responses`.

EXAMPLE 37: REUSABLE COMPONENTS

```
"components": {
  "schemas": …,
  "parameters": …,
  "responses": …
}
```

4.6.1 Field `schemas`

The value of `schemas` is a map of Schema Objects, see **[OpenAPI]**. Each entity type, complex type, enumeration type, and type definition directly or indirectly used in the `paths` field is represented as a name/value pair below the `schemas` map.

OData names can contain any Unicode letter or number, whereas keys for reusable objects in the Components Object must match the regular expression `^[a-zA-Z0-9\.\-_]+$`.

OData services whose schema and type names fulfil this restriction can directly place the name/value pairs for their types within the `schemas` map.

EXAMPLE 38: TYPE SCHEMAS WITHOUT INTERMEDIATE WRAPPER SCHEMA

```
"schemas": {
  "ODataDemo.Product": …,
  "ODataDemo.Category": …,
  "ODataDemo.Supplier": …,
  "ODataDemo.Country": …,
  "ODataDemo.Address": …,
  "org.example.Employee": …,
  "org.example.Manager": …
}
```

OData services using Unicode letters or numbers outside this restricted set have to use a dummy name/value pair in the `schemas` map whose name is e.g. `types`, and whose value is a schema of type `object` whose `properties` map then contains name/value pairs for the service's named types.

EXAMPLE 39: TYPE SCHEMAS WITH INTERMEDIATE WRAPPER SCHEMA

```
"schemas": {
  "types": {
    "type":"object",
    "properties": {
      "Liberté.Égalité": …,
      "ODataDemo.Product": …,
      "ODataDemo.Category": …,
      …
    }
  }
}
```

Of course, references to these types have to consider this intermediate wrapper schema:

EXAMPLE 40: TYPE REFERENCE WITH INTERMEDIATE WRAPPER SCHEMA

```
"$ref": "#/components/schemas/types/properties/Liberté.Égalité"
```

The name of each pair is the namespace-qualified name of the type. It uses the namespace instead of the alias because these definitions can be reused by other CSDL documents, and aliases are document-local, so they are meaningless for referencing documents.

The value of each pair is a Schema Object, see **[OpenAPI]**.

4.6.1.1 Schemas for Entity Types and Complex Types

A structured type is represented as a Schema Object of type `object`. The Schema Object contains the standard OpenAPI Specification keywords appropriate for type `object`.

It does not contain the `additionalProperties` keyword to allow additional properties beyond the declared properties. This is necessary for inheritance as well as instance annotations and dynamic properties.

It also does not contain the `required` keyword as response bodies can be arbitrarily tailored using `$select`, and `PATCH` request bodies can omit any unaltered properties.

It optionally can contain the field `title`, whose value is the value of the unqualified annotation `Core.Description` of the structured type, and the field `description`, whose value is the value of the unqualified annotation `Core.LongDescription`.

If desired, different Schema Objects can be used for creating, updating, and retrieving structured type instances. Schema Objects for create would omit all computed properties (see term `Core.Computed` in in **[OData-VocCore]**), Schema Objects for update would in addition omit all key properties and all immutable properties (see term `Core.Immutable` in **[OData-VocCore]**).

A structured type with a base type can be represented in several ways, depending on whether to preserve the aspect of inheriting properties of the base type, being able to substitute the derived type for its base type, or both.

Property inheritance is expressed with the keyword `allOf` whose value is an array with one item, a JSON Reference to the Schema Object of the base type. That way all properties of the referenced type are also valid for the current type, and this applies recursively.

Type substitutability is expressed with the keyword `anyOf` whose value is an array of JSON References to the Schema Objects of the known direct derived types. This also applies recursively, so only the direct derived types need to be represented.

Using both `allOf` referencing the base type's Schema Object and `anyOf` referencing the derived types' Schema Objects introduces cyclic references that derail many OpenAPI tools, so these two constructs should not be directly combined. If only the substitutability aspect (using `anyOf`) is necessary, the properties of base types (recursively) need to be copied into the Schema Object of a derived type.

A more complicated representation preserving both property inheritance and substitutability and avoiding cyclic references is to have two Schema Objects per structured type:

- A "property schema" defining the direct structural and navigation properties of the type and referencing the Schema Object of the base type with `allOf`, and
- A "payload schema" referencing the "property schema" with `allOf` and referencing the Schema Objects of the known direct derived types with `anyOf`, and not defining any properties. The "payload schema" is then referenced from Operation Objects.

This can be combined with having different Schema Objects for create, update, and read, leading to six Schema Objects per structured type.

EXAMPLE 41: PRODUCT ENTITY TYPE

```
"ODataDemo.Product": {
  "type": "object",
  "properties": {
    …
  },
  …
}
```

EXAMPLE 42: MANAGER ENTITY TYPE INHERITING FROM EMPLOYEE

```
"org.example.Manager": {
  "type": "object",
  "allOf": [
    {
      "$ref": "#/components/schemas/org.example.Employee"
    }
  ],
  "properties": {
    …
  },
  …
}
```

4.6.1.1 Properties

Each structural property and navigation property is represented as a name/value pair of the standard OpenAPI `properties` object. The name is the property name, the value is a Schema Object describing the allowed values of the property.

The Schema Object for a property optionally can contain the field `title`, whose value is the value of the unqualified annotation `Core.Description` of the property, and the field `description`, whose value is the value of the unqualified annotation `Core.LongDescription`.

EXAMPLE 43: STRUCTURAL AND NAVIGATION PROPERTIES OF SUPPLIER ENTITY TYPE

```
"ODataDemo.Supplier":{
  …,
  "properties":{
    "ID":…,
    "Name":…,
    "Address":…,
    "Concurrency":…,
    "Products":…
  },
  …
}
```

4.6.1.1.1 Primitive Properties

Primitive properties of type `Edm.PrimitiveType`, `Edm.Stream`, and any of the `Edm.Geo*` types are represented as Schema Objects that are JSON References to definitions in the Definitions Object or a reusable definitions file such as odata-definitions.json in the examples folder of the **[OData-OpenAPI]** OASIS GitHub repository.

All other primitive properties are represented as a Schema Object with the following OpenAPI Specification types, formats, and validation keywords:

OData Primitive Type	**OpenAPI Specification**			**Comment**
	Type	**Format**	**Keywords**	
`Edm.Binary`	`string`	`base64url`	`maxLength`	OData-specific format `maxLength` is the maximum length of the base64-encoded string representation, i.e. 4*ceil(`MaxLength`/3)
`Edm.Boolean`	`boolean`			
`Edm.Byte`	`integer`	`uint8`		OData-specific format
`Edm.Date`	`string`	`date`		OpenAPI format
`Edm.DateTimeOffset`	`string`	`date-time`		OpenAPI format
`Edm.Decimal`	`number`, `string`	`decimal`	`minimum` `maximum` `multipleOf`	OData-specific format `string` is only needed if the service supports the special values `-INF`, `INF`, and `NaN`
`Edm.Double`	`number`, `string`	`double`		OpenAPI format with extended meaning Type `string` is only needed if the service supports the special values `-INF`, `INF`, and `NaN` or the `IEEE754Compatible` format parameter
`Edm.Duration`	`string`	`duration`		OData-specific format
`Edm.Guid`	`string`	`uuid`		OData-specific format

OData Primitive Type	OpenAPI Specification			Comment
	Type	Format	Keywords	
`Edm.Int16`	`integer`	`int16`		OData-specific format
`Edm.Int32`	`integer`	`int32`		OpenAPI format
`Edm.Int64`	`integer`, `string`	`int64`		OpenAPI format Type `string` is only needed if the service supports the `IEEE754Compatible` format parameter
`Edm.SByte`	`integer`	`int8`		OData-specific format
`Edm.Single`	`number`, `string`	`float`		OpenAPI format with extended meaning Type `string` is only needed if the service supports the special values `-INF`, `INF`, and `NaN`
`Edm.String`	`string`		`maxLength`	Sequence of UTF-8 characters
`Edm.TimeOfDay`	`string`	`time`		OData-specific format

Properties of type `Edm.Decimal` and `Edm.Int64` are represented as JSON strings if the format option `IEEE754Compatible=true` is specified, so they have to be declared with both `number` and `string`, using an `anyOf` construct.

The scale of properties of type `Edm.Decimal` are represented with the OpenAPI Specification keyword `multipleOf` and a value of 10^{-scale}. The precision is represented with the `maximum` and `minimum` keywords and a value of $\pm(10^{precision-scale} - 10^{-scale})$ if the scale facet has a numeric value, and $\pm(10^{precision} - 1)$ if the scale is variable).

Properties of type `Edm.Double` and `Edm.Single` have special values for `-INF`, `INF`, and `NaN` that are represented as JSON strings, so they also have to be declared with both `number` and `string`. Services that do not support the special values `-INF`, `INF`, and `NaN` can just use type `number`.

Nullable properties are marked with the keyword `nullable` and a value of `true`.

The default value of a property is represented with the OpenAPI Specification keyword `default`.

Example 44: non-nullable Boolean property with default value

```
"BooleanValue": {
  "type": "boolean",
  "default": false
}
```

Example 45: non-nullable binary property with both maxLength and byteLength

```
"BinaryValue": {
  "type": "string",
  "format": "base64url",
  "maxLength": 44,
  "default":" T0RhdGE"
}
```

EXAMPLE 46: NON-NULLABLE INTEGER PROPERTY

```
"IntegerValue": {
  "type": "integer",
  "format": "int32",
  "default": -128
}
```

EXAMPLE 47: NON-NULLABLE FLOATING-POINT PROPERTIES: STRING REPRESENTATION FOR `-INF`, `INF`, AND `NaN`,

```
"DoubleValue": {
  "anyOf": [
    { "type": "number" },
    { "type": "string" }
  ],
  "format": "double",
  "default": 3.1415926535897931
},
"SingleValue": {
  "anyOf": [
    { "type": "number" },
    { "type": "string" }
  ],
  "format": "float"
}
```

EXAMPLE 48: NON-NULLABLE DECIMAL PROPERTY WITH UNSPECIFIED PRECISION: NO MINIMUM AND MAXIMUM

```
"DecimalValue": {
  "anyOf": [
    { "type": "number" },
    { "type": "string" }
  ],
  "format": "decimal",
  "default": 34.95
}
```

EXAMPLE 49: NON-NULLABLE DECIMAL PROPERTY WITH PRECISION 15 AND SCALE 2

```
"FixedDecimalValue": {
  "anyOf": [
    { "type": "number" },
    { "type": "string" }
  ],
  "format": "decimal",
  "multipleOf": 0.01,
  "minimum": -999999999.99,
  "maximum": 999999999.99
}
```

EXAMPLE 50: NULLABLE DECIMAL PROPERTY WITH PRECISION 15 AND SCALE 3

```
"NullableDecimalValue": {
  "anyOf": [
    { "type": "number" },
    { "type": "string"}
  ],
  "nullable": true
  "format":" decimal",
  "multipleOf": 1e-3,
  "minimum": -999999999999.999,
  "maximum": 999999999999.999
}
```

EXAMPLE 51: NON-NULLABLE STRING PROPERTY WITH MAXIMUM LENGTH OF 40 CHARACTERS

```
"StringValue": {
  "type": "string",
  "maxLength": 40
  "default":" Say \"Hello\",\nthen go"
}
```

EXAMPLE 52: NON-NULLABLE DATE PROPERTY

```
"DateValue": {
  "type": "string",
  "format": "date",
  "default": "2012-12-03"
}
```

EXAMPLE 53: NON-NULLABLE TIMESTAMP PROPERTY WITH 7 FRACTIONAL DIGITS PRECISION

```
"DateTimeOffsetValue": {
  "type": "string",
  "format": "date-time",
  "default": "2012-12-03T07:16:23:00.0000000Z"
}
```

EXAMPLE 54: NULLABLE TIMESTAMP PROPERTY

```
"NullableDateTimeOffsetValue": {
  "type": "string",
  "nullable": true,
  "format": "date-time"
}
```

EXAMPLE 55: NON-NULLABLE TIMESTAMP PROPERTY WITH 12 FRACTIONAL DIGITS PRECISION

```
"DurationValue": {
  "type": "string",
  "format": "duration",
  "default": "P12DT23H59M59.999999999999S"
}
```

EXAMPLE 56: NON-NULLABLE TIME PROPERTY WITH 3 FRACTIONAL DIGITS PRECISION

```
"TimeOfDayValue": {
  "type": "string",
  "format": "time",
  "default": "07:59:59.999"
}
```

EXAMPLE 57: NON-NULLABLE GUID PROPERTY WITH DEFAULT VALUE

```
"GuidValue": {
  "type": "string",
  "format": "uuid",
  "default": "1234567-89ab-cdef-0123-456789abcdef"
}
```

EXAMPLE 58: NON-NULLABLE 8-BYTE INTEGER PROPERTY, ALLOWING FOR STRING REPRESENTATION IN IEEE754COMPATIBLE MODE

```
"Int64Value": {
  "anyOf":[
    { "type": "integer" },
    { "type": "string" }
  ],
  "format": "int64",
  "default": 0
}
```

EXAMPLE 59: NON-NULLABLE ENUMERATION PROPERTY

```
"ColorEnumValue": {
  "anyOf": [
    {
      "$ref": "#/components/schemas/Model1.Color"
    }
  ],
  "default": "yellow"
},
```

EXAMPLE 60: NON-NULLABLE GEOGRAPHY-POINT PROPERTY

```
"GeographyPoint":{
  "anyOf": [
    {
      "$ref": "https://raw.githubusercontent.com/oasis-tcs/odata-
openapi/master/examples/odata-
definitions.json#/definitions/Edm.GeographyPoint"
    }
  ],
  "default": {
    "type": "Point",
    "coordinates": [
      142.1,
      64.1
    ]
  }
}
```

Example 61: non-nullable stream property: not part of payload in version 4.0

```
"StreamValue": {
  "$ref": "https://raw.githubusercontent.com/oasis-tcs/odata-
openapi/master/examples/odata-definitions.json#/definitions/Edm.Stream"
}
```

Example 62: non-nullable property typed with a type definition

```
"TypeDefValue": {
  "anyOf": [
    {
      "$ref": "#/components/schemas/Model1.IntegerDecimal"
    }
  ],
  "default": 42
}
```

Example 63: non-nullable primitive property with abstract type, e.g. in term definition

```
"PrimitiveValue": {
  "$ref": "https://raw.githubusercontent.com/oasis-tcs/odata-
openapi/master/examples/odata-definitions.json#/definitions/Edm.PrimitiveType"
}
```

4.6.1.1.2 Single-Value Complex and Navigation Properties

Complex properties are represented as JSON References to the Schema Object of the complex type, either as local references for types directly defined in the CSDL document, or as external references for types defined in referenced CSDL documents.

Navigation properties are represented similar to complex properties so that a standard OpenAPI Specification validator can validate the expanded representation of the navigation property.

Example 64: complex property Address

```
"Address": {
  "$ref":"#/components/schemas/ODataDemo.Address"
},
```

Example 65: single-valued navigation property Category

```
"Category": {
  "$ref": "#/components/schemas/ODataDemo.Category"
}
```

Example 66: nullable single-valued navigation property Country

```
"Country": {
  "nullable": true,
  "anyOf": [
    {
      "$ref": "#/components/schemas/ODataDemo.Category"
    }
  ]
}
```

4.6.1.1.3 Collection-Valued Properties

Collection-valued structural and navigation properties are represented as Schema Objects of type `array`. The value of the `items` keyword is a Schema Object specifying the type of the items.

EXAMPLE 67: COLLECTION-VALUED NAVIGATION PROPERTY PRODUCTS

```
"Products": {
  "type": "array",
  "items": {
    "$ref": "#/components/schemas/ODataDemo.Product"
  }
}
```

EXAMPLE 68: COLLECTION-VALUED COMPLEX PROPERTY ADDRESSES

```
"Adresses": {
  "type": "array",
  "items": {
    "$ref":"#/components/schemas/ODataDemo.Address"
  }
}
```

EXAMPLE 69: NULLABLE COLLECTION-VALUED PRIMITIVE PROPERTY DATES

```
"Dates": {
  "type": "array",
  "items": {
    "type": "string",
    "nullable": true,
    "format": "date"
  }
},
```

4.6.1.2 Schemas for Enumeration Types

An enumeration type is represented as a Schema Object of type `string` containing the OpenAPI Specification `enum` keyword. Its value is an array that contains a string with the member name for each enumeration member.

It optionally can contain the field `title`, whose value is the value of the unqualified annotation `Core.Description` of the enumeration type, and the field `description`, whose value is the value of the unqualified annotation `Core.LongDescription`.

EXAMPLE 70: ENUMERATION TYPE

```
"org.example.ShippingMethod": {
  "type": "string",
  "enum": [
    "FirstClass",
    "TwoDay",
    "Overnight"
  ]
}
```

4.6.1.3 Schemas for Type Definitions

A type definition is represented as a Schema Object describing the allowed values of the type definition using the same rules as described for primitive properties.

It optionally can contain the field `title`, whose value is the value of the unqualified annotation `Core.Description` of the type definition, and the field `description`, whose value is the value of the unqualified annotation `Core.LongDescription`.

EXAMPLE 71: TYPE DEFINITIONS BASED ON EDM.STRING, EDM.DECIMAL AND EDM.DATETIMEOFFSET

```
"Model1.Text50": {
  "type": "string",
  "maxLength": 50
},
"Model1.VariableDecimal": {
  "type": "number",
  "description": "A type definition"
},
"Model1.ExactTimestamp": {
  "type": "string",
  "format": "date-time"
}
```

4.6.2 Field `parameters`

The value of `parameters` is a map of Parameter Objects, see **[OpenAPI]**. It allows defining query options and headers that can be reused across operations of the service.

It contains one name/value pair per OData system query option supported by the service.

EXAMPLE 72: REUSABLE QUERY OPTIONS

```
"parameters": {
  "top": {
    "name": "$top",
    "schema": {
      "type": "integer"
    },
    "in": "query",
    "description": "Show only the first n items, see [OData Paging -
Top](http://docs.oasis-open.org/odata/odata/v4.01/odata-v4.01-part1-
protocol.html#sec_SystemQueryOptiontop)"
  },
  "skip": {
    "name": "$skip",
    "schema": {
      "type": "integer"
    },
    "in": "query",
    "description": "Skip the first n items, see [OData Paging -
Skip](http://docs.oasis-open.org/odata/odata/v4.01/odata-v4.01-part1-
protocol.html#sec_SystemQueryOptionskip)"
  },
  "count": {
    "name": "$count",
    "schema": {
      "type": "boolean"
    },
    "in": "query",
    "description": "Include count of items, see [OData
Count](http://docs.oasis-open.org/odata/odata/v4.01/odata-v4.01-part1-
protocol.html#sec_SystemQueryOptioncount)"
  },
  "filter": {
    "name": "$filter",
    "schema": {
      "type": "string"
    },
    "in": "query",
    "description": "Filter items by property values, see [OData
Filtering](http://docs.oasis-open.org/odata/odata/v4.01/odata-v4.01-part1-
protocol.html#sec_SystemQueryOptionfilter)"
  },
  "search": {
    "name": "$search",
    "schema": {
      "type": "string"
    },
    "in": "query",
    "description": "Search items by search phrases, see [OData
Searching](http://docs.oasis-open.org/odata/odata/v4.01/odata-v4.01-part1-
protocol.html#sec_SystemQueryOptionsearch)"
  }
}
```

4.6.3 Field `responses`

The value of `responses` is a map of Response Objects, see **[OpenAPI]**. It allows defining responses that can be reused across operations of the service.

It contains one name/value pair for the standard OData error response that is referenced from all operations of the service. The reusable error response in turn references a Schema Object in `schemas` or a reusable definitions file such as odata-definitions.json in the examples folder of the **[OData-OpenAPI]** OASIS GitHub repository.

EXAMPLE 73: REUSABLE ERROR RESPONSE

```
"responses": {
  "error": {
    "description": "Error",
    "content": {
      "application/json": {
        "schema": {
          "$ref": "#/components/schemas/odata.error"
        }
      }
    }
  }
}
```

4.6.4 Field `securitySchemes`

The value of `securitySchemes` is a map of Security Scheme Objects, see **[OpenAPI]**. It allows defining security schemes that can be reused across operations of the service.

The Security Scheme Objects can be constructed from items of annotations with the term `Authorization.Authorizations` in **[OData-VocAuth]**.

4.7 Field security

The value of `security` is an array of Security Requirement Objects, see **[OpenAPI]**. It lists the required security schemes to execute operations of the service.

The Security Requirements Objects can be constructed from items of annotations with the term `Authorization.SecuritySchemes` in **[OData-VocAuth]**.

5 Annotations influencing OAS Documents

Annotations can influence the contents of an OAS document generated from an OData metadata document.

5.1 Authorization Vocabulary

The following terms from **[OData-VocAuth]** can be applied to the entity container:

Term	OpenAPI field
`Authorizations`	`securitySchemes` of Components Object
`SecuritySchemes`	`security` of OpenAPI Object

5.2 Capabilities Vocabulary

The following terms from **[OData-VocCap]** can be applied to the entity container:

Term	OpenAPI field
`KeyAsSegmentSupported`	Path templates for key access

The following terms from **[OData-VocCap]** can be applied to entity sets and singletons:

Term	OpenAPI field
`DeleteRestrictions`	Operation Object for `delete`
`ExpandRestrictions`	`parameters` of Operation Object for `get`
`NavigationRestrictions`	Paths for navigation properties
`ReadRestrictions`	Operation Object for `get`
`SelectSupport`	Operation Object for `get`
`UpdateRestrictions`	Operation Object for `patch`

The following terms from **[OData-VocCap]** can be applied to entity sets:

Term	OpenAPI field
`CountRestrictions`	`parameters` of Operation Object for `get`
`FilterRestrictions`	`parameters` of Operation Object for `get`
`IndexableByKey`	Paths for key access
`InsertRestrictions`	Operation Object for `post`
`ReadByKeyRestrictions`	Operation Object for `get`
`SearchRestrictions`	`parameters` of Operation Object for `get`
`SkipSupported`	`parameters` of Operation Object for `get`
`SortRestrictions`	`parameters` of Operation Object for `get`
`TopSupported`	`parameters` of Operation Object for `get`

5.3 Core Vocabulary

The following terms from **[OData-VocCore]** can be applied to any model element:

Term	OpenAPI field
`Description`	`title` of Info Object when applied to a schema or entity container `title` of Schema Object for a property or type `summary` of Operation Object for an action or function import `summary` of Operation Object for a bound action or function `description` of a parameter object
`LongDescription`	`description` of Info Object when applied to a schema or entity container `description` of Schema Object for a property or type `description` of Operation Object for an action or function import `description` of Operation Object for a bound action or function

The following terms from **[OData-VocCore]** can be applied to a schema:

Term	OpenAPI field
`DefaultNamespace`	Path templates for bound actions and functions
`SchemaVersion`	`version` of Info Object

The following terms from **[OData-VocCore]** can be applied to properties:

Term	OpenAPI field
`Computed`	omit from Schema Object for create and update requests
`Immutable`	omit from Schema Object for update requests
`Example`	`example` of Schema Object

5.4 Validation Vocabulary

The following terms from **[OData-VocVal]** can be applied to primitive properties:

Term	OpenAPI field
`AllowedValues`	`enum` of Schema Object
`Exclusive`	`exclusiveMinimum` / `exclusiveMaximum` of Schema Object
`Maximum`	`maximum` of Schema Object
`Minimum`	`minimum` of Schema Object
`Pattern`	`pattern` of Schema Object

6 Example

The Products and Categories example metadata document in section 16.1 of **[OData-CSDL]** is available in machine-readable form at https://oasis-tcs.github.io/odata-openapi/examples/csdl-16.1.xml.

Its OAS counterpart is available at https://oasis-tcs.github.io/odata-openapi/examples/csdl-16.1.openapi3.json.

Appendix A. Acknowledgments

The contributions of the OASIS OData Technical Committee members, enumerated in **[OData-Protocol]**, are gratefully acknowledged.

Appendix B. Revision History

Revision	Date	Editor	Changes Made
Committee Note Draft 01	2016-11-28	Ralf Handl	Initial version
Committee Note Draft 02	2019-07-05	Ralf Handl	Based on OpenAPI 3.0.2