OData JSON Format Version 4.02

Committee Specification Draft 01

14 July 2023

This stage:

https://docs.oasis-open.org/odata/odata-json-format/v4.02/csd01/odata-json-format-v4.02-csd01.md (Authoritative)
https://docs.oasis-open.org/odata/odata-json-format/v4.02/csd01/odata-json-format-v4.02-csd01.html
https://docs.oasis-open.org/odata/odata-json-format/v4.02/csd01/odata-json-format-v4.02-csd01.pdf

Previous stage:

N/A

Latest stage:

https://docs.oasis-open.org/odata/odata-json-format/v4.02/odata-json-format-v4.02.md (Authoritative)
https://docs.oasis-open.org/odata/odata-json-format/v4.02/odata-json-format-v4.02.html
https://docs.oasis-open.org/odata/odata-json-format/v4.02/odata-json-format-v4.02.pdf

Technical Committee:

OASIS Open Data Protocol (OData) TC

Chairs:

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

Editors:

Ralf Handl (ralf.handl@sap.com), SAP SE
Michael Pizzo (mikep@microsoft.com), Microsoft
Heiko Theißen (heiko.theissen@sap.com), SAP SE

Related work:

This specification replaces or supersedes:

- OData JSON Format Version 4.01. Edited by Michael Pizzo, Ralf Handl, and Mark Biamonte. OASIS Standard. Latest stage: https://docs.oasis-open.org/odata/odata-json-

format/v4.01/odata-json-format-v4.01.html.

- OData JSON Format Version 4.0. Edited by Ralf Handl, Michael Pizzo, and Mark Biamonte. OASIS Standard. Latest stage: http://docs.oasis-open.org/odata/odata-json-format/v4.0/odata-json-format-v4.0.html.

This specification is related to:

- *OData Version 4.02*. Edited by Michael Pizzo, Ralf Handl, and Heiko Theißen. A multi-part Work Product that includes:
  - *OData Version 4.02 Part 1: Protocol*. Latest stage. https://docs.oasis-open.org/odata/odata/v4.02/odata-v4.02-part1-protocol.html
  - *OData Version 4.02 Part 2: URL Conventions*. Latest stage. https://docs.oasis-open.org/odata/odata/v4.02/odata-v4.02-part2-url-conventions.html
  - *ABNF components: OData ABNF Construction Rules Version 4.02 and OData ABNF Test Cases*. https://docs.oasis-open.org/odata/odata/v4.02/csd01/abnf/
- *OData Vocabularies Version 4.0*. Edited by Michael Pizzo, Ralf Handl, and Ram Jeyaraman. Latest stage: https://docs.oasis-open.org/odata/odata-vocabularies/v4.0/odata-vocabularies-v4.0.html
- *OData Common Schema Definition Language (CSDL) JSON Representation Version 4.02*. Edited by Michael Pizzo, Ralf Handl, and Heiko Theißen. Latest stage: https://docs.oasis-open.org/odata/odata-csdl-json/v4.02/odata-csdl-json-v4.02.html
- *OData Common Schema Definition Language (CSDL) XML Representation Version 4.02*. Edited by Michael Pizzo, Ralf Handl, and Heiko Theißen. Latest stage: https://docs.oasis-open.org/odata/odata-csdl-xml/v4.02/odata-csdl-xml-v4.02.html

Abstract:

The Open Data Protocol (OData) for representing and interacting with structured content is comprised of a set of specifications. The core specification for the protocol is in OData Version 4.02 Part 1: Protocol. This document extends the core specification by defining representations for OData requests and responses using a JSON format.

Status:

This document was last revised or approved by the OASIS Open Data Protocol (OData) TC on the above date. The level of approval is also listed above. Check the "Latest stage" location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=odata#technical.

TC members should send comments on this specification to the TC's email list. Others should send comments to the TC's public comment list, after subscribing to it by following the instructions at the "Send A Comment" button on the TC's web page at https://www.oasis-open.org/committees/odata/.

This specification is provided under the RF on RAND Terms Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC's web page (https://www.oasis-open.org/committees/odata/ipr.php).

Note that any machine-readable content (Computer Language Definitions) declared Normative for this Work Product is provided in separate plain text files. In the event of a discrepancy between any such plain text file and display content in the Work Product's prose narrative document(s), the content in the separate plain text file prevails.

Key words:

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

Citation format:

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

[OData-JSON-Format-v4.02]

OData JSON Format Version 4.02. Edited by Ralf Handl, Michael Pizzo, and Heiko Theißen. 14 July 2023. OASIS Committee Specification Draft 01. https://docs.oasis-open.org/odata/odata-json-format/v4.02/csd01/odata-json-format-v4.02-csd01.html. Latest stage: https://docs.oasis-open.org/odata/odata-json-format/v4.02/odata-json-format-v4.02.html.

Notices

Table of Contents

1 Introduction

The OData protocol is comprised of a set of specifications for representing and interacting with structured content. The core specification for the protocol is in **[OData-Protocol]**; this document is an extension of the core protocol. This document defines representations for the OData requests and responses using the JavaScript Object Notation (JSON), see [RFC8259].

An OData JSON payload may represent:

- a single primitive value
- a collection of primitive values
- a single complex type value
- a collection of complex type values
- a single entity or entity reference
- a collection of entities or entity references
- a collection of changes
- a service document describing the top-level resources exposed by the service
- an error.

1.1 Changes from earlier Versions

1.2 Glossary

1.2.1 Definitions of terms

TODO: find out why we need a $dummy$ formula to get `monospace` look as we want it.

1.2.2 Acronyms and abbreviations

1.2.3 Document conventions

Keywords defined by this specification use `this monospaced font`.

Some sections of this specification are illustrated with non-normative examples.

Example 1: text describing an example uses this paragraph style

```
Non-normative examples use this paragraph style.
```

All examples in this document are non-normative and informative only. Examples labeled with ⚠ contain advanced concepts or make use of keywords that are defined only later in the text, they can be skipped at first reading.

All other text is normative unless otherwise labeled.

Here is a customized command line which will generate HTML from this markdown file (named `odata-json-format-v4.02-csd01.md`). Line breaks are added for readability only:

```
pandoc -f gfm+tex_math_dollars+fenced_divs
       -t html
```



```
      -o odata-json-format-v4.02-csd01.html
      -c styles/markdown-styles-v1.7.3b.css
      -c styles/odata.css
      -s
      --mathjax
      --eol=lf
      --wrap=none
      --metadata pagetitle="OData JSON Format Version 4.02"
      odata-json-format-v4.02-csd01.md
```

This uses pandoc 3.1.2 from https://github.com/jgm/pandoc/releases/tag/3.1.2.

2 JSON Format Design

JSON, as described in **[RFC8259]** defines a text format for serializing structured data. Objects are serialized as an unordered collection of name/value pairs.

JSON does not define any semantics around the name/value pairs that make up an object, nor does it define an extensibility mechanism for adding control information to a payload.

OData's JSON format extends JSON by defining general conventions for name/value pairs that annotate a JSON object, property or array. OData defines a set of canonical name/value pairs for control information such as ids, types, and links, and instance annotations MAY be used to add domain-specific information to the payload.

A key feature of OData's JSON format is to allow omitting predictable parts of the wire format from the actual payload. To reconstitute this data on the receiving end, expressions are used to compute missing links, type information, and other control data. These expressions (together with the data on the wire) can be used by the client to compute predictable payload pieces as if they had been included on the wire directly.

Control information is used in JSON to capture instance metadata that cannot be predicted (e.g. the next link of a collection) as well as a mechanism to provide values where a computed value would be wrong (e.g. if the media read link of one particular entity does not follow the standard URL conventions). Computing values from metadata expressions is compute intensive and some clients might opt for a larger payload size to avoid computational complexity; to accommodate for this the `Accept` header allows the client to control the amount of control information added to the response.

To optimize streaming scenarios, there are a few restrictions that MAY be imposed on the sequence in which name/value pairs appear within JSON objects. For details on the ordering requirements see Payload Ordering Constraints.

3 Requesting the JSON Format

The OData JSON format can be requested using the `$format` query option in the request URL with the media type `application/json`, optionally followed by format parameters, or the case-insensitive abbreviation `json` which MUST NOT be followed by format parameters.

Alternatively, this format can be requested using the `Accept` header with the media type `application/json`, optionally followed by format parameters.

If specified, `$format` overrides any value specified in the `Accept` header.

Possible format parameters are:

- `ExponentialDecimals`
- `IEEE754Compatible`
- `metadata`
- `streaming`

The names and values of these format parameters are case-insensitive.

Services SHOULD advertise the supported media types by annotating the entity container with the term `Capabilities.SupportedFormats` defined in **[OData-VocCap]**, listing all available formats and combinations of supported format parameters.

3.1 Controlling the Amount of Control Information in Responses

The amount of control information needed (or desired) in the payload depends on the client application and device. The `metadata` parameter can be applied to the `Accept` header of an OData request to influence how much control information will be included in the response.

Other `Accept` header parameters (e.g., `streaming`) are orthogonal to the `metadata` parameter and are therefore not mentioned in this section.

If a client prefers a very small wire size and is intelligent enough to compute data using metadata expressions, the `Accept` header should include `metadata=minimal`. If computation is more critical than wire size or the client is incapable of computing control information, `metadata=full` directs the service to inline the control information that normally would be computed from metadata expressions in the payload. `metadata=none` is an option for clients that have out-of-band knowledge or don't require control information.

In addition, the client may use the `include-annotations` preference in the `Prefer` header to request additional control information. Services supporting this MUST NOT omit control information required by the chosen `metadata` parameter, and services MUST NOT exclude the `nextLink`, `deltaLink`, and `count` if they are required by the response type.

If the client includes the `OData-MaxVersion` header in a request and does not specify the `metadata` format parameter in either the `Accept` header or `$format` query option, the service MUST return at least the minimal control information.

Note that in OData 4.0 the `metadata` format parameter was prefixed with `odata.`. Payloads with an `OData-Version` header equal to `4.0` MUST include the `odata.` prefix. Payloads with an `OData-Version` header equal to `4.01` or greater SHOULD NOT include the `odata.` prefix.

3.1.1 metadata=minimal (odata.metadata=minimal)

The `metadata=minimal` format parameter indicates that the service SHOULD remove computable control information from the payload wherever possible. The response payload MUST contain at least the following control information:

- `context`: the root context URL of the payload and the context URL for any deleted entries or added or deleted links in a delta response, or for entities or entity collections whose set cannot be determined from the root context URL
- `etag`: the ETag of the entity or collection, as appropriate
- `count`: the total count of a collection of entities or collection of entity references, if requested
- `nextLink`: the next link of a collection with partial results
- `deltaLink`: the delta link for obtaining changes to the result, if requested

In addition, control information MUST appear in the payload for cases where actual values are not the same as the computed values and MAY appear otherwise. When control information appears in the payload, it is treated as exceptions to the computed values.

Media entities and stream properties MAY in addition contain the following control information:

- `mediaEtag`: the ETag of the stream, as appropriate
- `mediaContentType`: the media type of the stream

3.1.2 metadata=full (odata.metadata=full)

The `metadata=full` format parameter indicates that the service MUST include all control information explicitly in the payload.

The full list of control information that may appear in a `metadata=full` response is as follows:

- `context`: the context URL for a collection, entity, primitive value, or service document.
- `count`: the total count of a collection of entities or collection of entity references, if requested.
- `nextLink`: the next link of a collection with partial results
- `deltaLink`: the delta link for obtaining changes to the result, if requested
- `id`: the ID of the entity
- `etag`: the ETag of the entity or collection, as appropriate
- `readLink`: the link used to read the entity, if the edit link cannot be used to read the entity
- `editLink`: the link used to edit/update the entity, if the entity is updatable and the `id` does not represent a URL that can be used to edit the entity
- `navigationLink`: the link used to retrieve the values of a navigation property
- `associationLink`: the link used to describe the relationship between this entity and related entities
- `type`: the type of the containing object or targeted property if the type of the object or targeted property cannot be heuristically determined from the data value, see section Control

Information: type (odata.type).

Media entities and stream properties may in addition contain the following control information:

- mediaReadLink: the link used to read the stream
- mediaEditLink: the link used to edit/update the stream
- mediaEtag: the ETag of the stream, as appropriate
- mediaContentType: the media type of the stream

3.1.3 metadata=none (odata.metadata=none)

The `metadata=none` format parameter indicates that the service SHOULD omit control information other than `nextLink` and `count`. This control information MUST continue to be included, as applicable, even in the `metadata=none` case.

It is not valid to specify `metadata=none` on a delta request.

3.2 Controlling the Representation of Numbers

The `IEEE754Compatible=true` format parameter indicates that the service MUST serialize `Edm.Int64` and `Edm.Decimal` numbers (including the `count`, if requested) as strings. This is in conformance with **[RFC7493]**.

If not specified, or specified as `IEEE754Compatible=false`, all numbers MUST be serialized as JSON numbers.

This enables support for JavaScript numbers that are defined to be 64-bit binary format IEEE 754 values (see **[ECMAScript, section 4.3.1.9]**) resulting in integers losing precision past 15 digits, and decimals losing precision due to the conversion from base 10 to base 2.

OData JSON request and response payloads that format `Edm.Int64` and `Edm.Decimal` values as strings MUST specify this format parameter in the media type sent in the `Content-Type` header.

Services producing responses without format parameter `IEEE754Compatible=true` which are unable to produce exact JSON numbers MAY serialize `Edm.Int64` and `Edm.Decimal` numbers with a rounded/inexact value as a JSON number and annotate that value with an instance annotation with term `Core.ValueException` defined in **[OData-VocCore]** containing the exact value as a string. This situation can for example happen if the client only accepts `application/json` without any format parameters and the service is written in JavaScript.

For payloads with an `OData-Version` header equal to `4.0` the `ExponentialDecimals=true` format parameter indicates that the service MAY serialize `Edm.Decimal` numbers in exponential notation (e.g. `1e-6` instead of `0.000001`).

The sender of a request MUST specify `ExponentialDecimals=true` in the `Content-Type` header if the request body contains `Edm.Decimal` values in exponential notation.

If not specified, or specified as `ExponentialDecimals=false`, all `Edm.Decimal` values MUST be serialized in long notation, using only an optional sign, digits, and an optional decimal point followed by digits.

Payloads with an `OData-Version` header equal to `4.01` or greater always allow exponential notation for numbers and the `ExponentialDecimals` format parameter is not needed or used.

4 Common Characteristics

This section describes common characteristics of the representation for OData values in JSON. A request or response body consists of several parts. It contains OData values as part of a larger document. Requests and responses are structured almost identical; the few existing differences will be explicitly called out in the respective subsections.

4.1 Header Content-Type

Requests and responses with a JSON message body MUST have a `Content-Type` header value of `application/json`.

Requests MAY add the `charset` parameter to the content type. Allowed values are `UTF-8`, `UTF-16`, and `UTF-32`. If no `charset` parameter is present, `UTF-8` MUST be assumed.

Responses MUST include the `metadata` parameter to specify the amount of metadata included in the response.

Requests and responses MUST include the `IEEE754Compatible` parameter if `Edm.Int64` and `Edm.Decimal` numbers are represented as strings.

Requests and responses MAY add the `streaming` parameter with a value of `true` or `false`, see section Payload Ordering Constraints.

4.2 Message Body

Each message body is represented as a single JSON object. This object is either the representation of an entity, an entity reference or a complex type instance, or it contains a name/value pair whose name MUST be `value` and whose value is the correct representation for a primitive value, a collection of primitive values, a collection of complex values, a collection of entities, or a collection of objects that represent changes to a previous result.

Client libraries MUST retain the order of objects within an array in JSON responses.

4.3 Relative URLs

URLs present in a payload (whether request or response) MAY be represented as relative URLs.

Relative URLs, other than those in `type`, are relative to their base URL, which is

- the context URL of the same JSON object, if one exists, otherwise
- the context URL of the enclosing object, if one exists, otherwise
- the context URL of the next enclosing object, if one exists, etc. until the document root, otherwise
- the request URL.

For context URLs, these rules apply starting with the second bullet point.

Within the `type` control information, relative URLs are relative to the base type URL, which is

- the `type` of the enclosing object, if one exists, otherwise
- the `type` of the next enclosing object, if one exists, etc. until the document root, otherwise
- the context URL of the document root, if one exists, otherwise
- the request URL.

Processors expanding the URLs MUST use normal URL expansion rules as defined in **[RFC3986]**. This means that if the base URL is a context URL, the part starting with `$metadata#` is ignored when resolving the relative URL.

Clients that receive relative URLs in response payloads SHOULD use the same relative URLs, where appropriate, in request payloads (such as bind operations and batch requests) and in system query options (such as `$id`).

URLs represented as a string within a JSON payload, including batch requests, must follow standard OData encoding rules. For relative URLs this means that colons in the path part, especially within key values, MUST be percent-encoded to avoid confusion with the scheme separator. Colons within the query part, i.e. after the question mark character (`?`), need not be percent-encoded.

Example 2:

```
{
  "@context": "http://host/service/$metadata#Customers/$entity",
  ...
  "@editLink": "Customers('ALFKI')",
  ...
  "Orders@navigationLink": "Customers('ALFKI')/Orders",
  ...
}
```

The resulting absolute URLs are `http://host/service/Customers('ALFKI')` and `http://host/service/Customers('ALFKI')/Orders`.

4.4 Payload Ordering Constraints

Ordering constraints MAY be imposed on the JSON payload in order to support streaming scenarios. These ordering constraints MUST only be assumed if explicitly specified as some clients (and services) might not be able to control, or might not care about, the order of the JSON properties in the payload.

Clients can request that a JSON response conform to these ordering constraints by specifying a media type of [application/json]{style="font-family: "Courier New""} with the `streaming=true` parameter in the `Accept` header or `$format` query option. Services MUST return `406 Not Acceptable` if the client only requests streaming and the service does not support it.

Clients may specify the `streaming=true` parameter in the `Content-Type` header of requests to indicate that the request body follows the payload ordering constraints. In the absence of this parameter, the service must assume that the JSON properties in the request are unordered.

Processors MUST only assume streaming support if it is explicitly indicated in the `Content-Type` header via the `streaming=true` parameter.

Example 3: a payload with

```
Content-Type: application/json;metadata=minimal;streaming=true
```

can be assumed to support streaming, whereas a payload with

```
Content-Type: application/json;metadata=minimal
```

cannot be assumed to support streaming.

JSON producers are encouraged to follow the payload ordering constraints whenever possible (and include the `streaming=true` content-type parameter) to support the maximum set of client scenarios.

To support streaming scenarios the following payload ordering constraints have to be met:

- If present, the `context` control information MUST be the first property in the JSON object.
- The `type` control information, if present, MUST appear next in the JSON object.
- The `id` and `etag` control information MUST appear before any property, property annotation, or property control information.
- All annotations or control information for a structural or navigation property MUST appear as a group immediately before the property itself. The one exception is the `nextLink` of a collection which MAY appear after the collection it annotates.
- All other control information can appear anywhere in the payload as long as it does not violate any of the above rules.
- For 4.0 payloads, annotations and control information for navigation properties MUST appear after all structural properties. 4.01 clients MUST NOT assume this ordering.

Note that in OData 4.0 the `streaming` format parameter was prefixed with `odata.`. Payloads with an `OData-Version` header equal to `4.0` MUST include the `odata.` prefix. Payloads with an `OData-Version` header equal to `4.01` or greater SHOULD NOT include the `odata.` prefix.

4.5 Control Information

In addition to the "pure data" a message body MAY contain annotations and control information that is represented as name/value pairs whose names start with `@`.

In requests and responses with an `OData-Version` header with a value of `4.0` control information names are prefixed with `@odata.`, e.g. `@odata.context`. In requests and responses without such a header the `odata.` prefix SHOULD be omitted, e.g `@context`.

In some cases, control information is required in request payloads; this is called out in the following subsections.

Receivers that encounter unknown annotations in any namespace or unknown control information MUST NOT stop processing and MUST NOT signal an error.

4.5.1 Control Information: context (odata.context)

The `context` control information returns the context URL (see **[OData-Protocol]**) for the payload. This URL can be absolute or relative.

The `context` control information is not returned if `metadata=none`[ ]{.MsoHyperlink}is requested. Otherwise[ ]{.MsoHyperlink}it MUST be the first property of any JSON response[.]{.MsoHyperlink}

The `context` control information MUST also be included in requests and responses for entities whose entity set cannot be determined from the context URL[ ]{.MsoHyperlink}of the collection.

For more information on the format of the context URL, see **[OData-Protocol]**.

Request payloads MAY include a context URL as a base URL for relative URLs in the request payload.

Example 4:

```
{
  "@context": "http://host/service/$metadata#Customers/$entity",
  "@metadataEtag": "W/\"A1FF3E230954908F\"",
  ...
}
```

4.5.2 Control Information: metadataEtag (odata.metadataEtag)

The `metadataEtag` control information MAY appear in a response in order to specify the entity tag (ETag) that can be used to determine the version of the metadata of the response. If an ETag is returned when requesting the metadata document, then the service SHOULD set the `metadataEtag` control information to the metadata document's ETag in all responses when using `metadata=minimal` or `metadata=full`. If no ETag is returned when requesting the metadata document, then the service SHOULD NOT set the `metadataEtag` control information in any responses.

For details on how ETags are used, see **[OData-Protocol]**.

4.5.3 Control Information: type (odata.type)

The `type` control information specifies the type of a JSON object or name/value pair. Its value is a URI that identifies the type of the property or object. For built-in primitive types the value is the unqualified name of the primitive type. For payloads described by an `OData-Version` header with a value of `4.0`, this name MUST be prefixed with the hash symbol (`#`); for non-OData 4.0 payloads, built-in primitive type values SHOULD be represented without the hash symbol, but consumers of 4.01 or greater payloads MUST support values with or without the hash symbol. For all other types, the URI may be absolute or relative to the `type` of the containing object. The root `type` may be absolute or relative to the root context URL.

If the URI references a metadata document (that is, it's not just a fragment), it MAY refer to a specific version of that metadata document using the `$schemaversion` system query option defined in **[OData-Protocol]**.

For non-built in primitive types, the URI contains the namespace-qualified or alias-qualified type, specified as a URI fragment. For properties that represent a collection of values, the fragment is the namespace-qualified or alias-qualified element type enclosed in parentheses and prefixed with `Collection`. The namespace or alias MUST be defined or the namespace referenced in the metadata document of the service, see **[OData-CSDLJSON]** or **[OData-CSDLXML]**.

The `type` control information MUST appear in requests and in responses with minimal or full metadata, if the type cannot be heuristically determined, as described below, and one of the following is true:

- The type is derived from the type specified for the (collection of) entities or (collection of) complex type instances, or
- The type is for a property whose type is not declared in `$metadata`.

The following heuristics are used to determine the primitive type of a dynamic property in the absence of the `type` control information:

- Boolean values have a first-class representation in JSON and do not need any additional control information.
- Numeric values have a first-class representation in JSON but are not further distinguished, so they include a `type` control information unless their type is `Double`.
- The special floating-point values `-INF`, `INF`, and `NaN` are serialized as strings and MUST have a `type` control information to specify the numeric type of the property.
- String values do have a first class representation in JSON, but there is an obvious collision: OData also encodes a number of other primitive types as strings, e.g. `DateTimeOffset`, `Int64` in the presence of the `IEEE754Compatible` format parameter etc. If a property appears in JSON string format, it should be treated as a string value unless the property is known (from the metadata document) to have a different type.

For more information on namespace- and alias-qualified names, see **[OData-CSDLJSON]** or **[OData-CSDLXML]**.

Example 5: entity of type `Model.VipCustomer` *defined in the metadata document of the same service with a dynamic property of type* `Edm.Date`

```
{
  "@context": "http://host/service/$metadata#Customers/$entity",
  "@type": "#Model.VipCustomer",
  "ID": 2,
  "DynamicValue@type": "Date",
  "DynamicValue": "2016-09-22",
  ...
}
```

Example 6: entity of type `Model.VipCustomer` *defined in the metadata document of a different service*

```
{
  "@context": "http://host/service/$metadata#Customers/$entity",
  "@type": "http://host/alternate/$metadata#Model.VipCustomer",
  "ID": 2,
```



```
  ...
}
```


5 Service Document

6 Entity

9 Stream Property

10 Media Entity

11 Individual Property or Operation Response

12 Collection of Operation Responses

13 Collection of Entities

14 Entity Reference

16 Bound Function

17 Bound Action

18 Action Invocation

20 Instance Annotations

20.1 Annotate a JSON Object

20.2 Annotate a JSON Array or Primitive

20.3 Annotate a Primitive Value within a JSON Array

22 Extensibility

23 Conformance

Conforming clients MUST be prepared to consume a service that uses any or all of the constructs defined in this specification. The exception to this are the constructs defined in Delta Response, which are only required for clients that request changes.

In order to be a conforming consumer of the OData JSON format, a client or service:

1. MUST either:
    1. understand `metadata=minimal` (section 3.1.1) or
    2. explicitly specify `metadata=none` (section 3.1.3) or `metadata=full` (section 3.1.2) in the request (client)
2. MUST be prepared to consume a response with full metadata
3. MUST be prepared to receive all data types (section 7.1)
    1. defined in this specification (client)
    2. exposed by the service (service)
4. MUST interpret all `odata` control information defined according to the `OData-Version` header of the payload (section 4.5)
5. MUST be prepared to receive any annotations and control information not defined in the `OData-Version` header of the payload (section 20)
6. MUST NOT require `streaming=true` in the `Content-Type` header (section 4.4)
7. MUST be a conforming consumer of the OData 4.0 JSON format, for payloads with an `OData-Version` header value of `4.0`.
    1. MUST accept the `odata.` prefix, where defined, on format parameters and control information
    2. MUST accept the `#` prefix in `@odata.type` values
    3. MUST be prepared to handle binding through the use of the `@odata.bind` property in payloads to a `PATCH`, `PUT`, or `POST` request
    4. MUST accept `TargetId` within in a deleted link for a relationship with a maximum cardinality of one
    5. MUST accept the string values `-INF`, `INF`, and `NaN` for single and double values
    6. MUST support property annotations that appear immediately before or after the property they annotate
8. MAY be a conforming consumer of the OData 4.01 JSON format, for payloads with an `OData-Version` header value of `4.01`.
    1. MUST be prepared to interpret control information with or without the `odata.` prefix
    2. MUST be prepared for `@odata.type` primitive values with or without the `#` prefix
    3. MUST be prepared to handle binding through inclusion of an entity reference within a collection-valued navigation property in the body of a `PATCH`, `PUT`, or `POST` request
    4. MUST be prepared for `TargetId` to be included or omitted in a deleted link for a relationship with a maximum cardinality of one
    5. MUST accept the string values `-INF`, `INF`, and `NaN` for decimal values with floating scale
    6. MUST be prepared to handle related entities inline within a delta payload as well as a nested delta representation for the collection

7. MUST be prepared to handle decimal values written in exponential notation

In order to be a conforming producer of the OData JSON format, a client or service:

9. MUST support generating OData 4.0 JSON compliant payloads with an `OData-Version` header value of `4.0`.
    1. MUST NOT omit the `odata.` prefix from format parameters or control information
    2. MUST NOT omit the `#` prefix from `@odata.type` values
    3. MUST NOT include entity values or entity references within a collection-valued navigation property in the body of a `PATCH`, `PUT`, or `POST` request
    4. MUST NOT return decimal values written in exponential notation unless the ExponentialDecimals format parameter is specified.
    5. MUST NOT advertise available actions or functions using name/value pairs prefixed with a property name
    6. MUST NOT return a null value for name/value pairs representing actions or functions that are not available
    7. MUST NOT represent numeric value exceptions for values other than single and double values using the string values `-INF`, `INF`, and `NaN`
10. MAY support generating OData 4.01 JSON compliant payloads for requests with an `OData-Version` header value of `4.01`.
    1. MUST return property annotations immediately before the property they annotate
    2. SHOULD omit the `odata.` prefix from format parameters and control information
    3. SHOULD omit the `#` prefix from `@type` primitive values
    4. MAY include inline related entities or nested delta collections within a delta payload
    5. MAY include `TargetId` within a deleted link for a relationship with a maximum cardinality of 1
    6. MAY return decimal values written in exponential notation
    7. MAY represent numeric value exceptions for decimal values with floating scale using the string values `-INF`, `INF`, and `NaN`

In addition, in order to conform to the OData JSON format, a service:

11. MUST comply with one of the conformance levels defined in **[OData-Protocol]**
12. MUST support the `application/json` media type in the `Accept` header (section 3)
13. MUST return well-formed JSON payloads
14. MUST support `odata.metadata=full` (section 3.1.2)
15. MUST include the `odata.nextLink` control information in partial results for entity collections (section 4.5.5)
16. MUST support entity instances with external metadata (section 4.5.1)
17. MUST support properties with externally defined data types (section 4.5.3)
18. MUST NOT violate any other aspects of this OData JSON specification
19. SHOULD support the `$format` system query option (section 3)
20. MAY support the `odata.streaming=true` parameter in the `Accept` header (section 4.4)
21. MAY return full metadata regardless of `odata.metadata` (section 3.1.2)

22. MUST NOT omit null or default values unless the `omit-values` preference is specified in the `Prefer` request header and the `omit-values` preference is included in the `Preference-Applied` response header
23. MUST return OData JSON 4.0-compliant responses for requests with an `OData-MaxVersion` header value of `4.0`
24. MUST support OData JSON 4.0-compliant payloads in requests with an `OData-Version` header value of `4.0`
25. MUST support returning, in the final response to an asynchronous request, the `application/json` payload that would have been returned had the operation completed synchronously, wrapped in an `application/http` message

In addition, in order to comply with the OData 4.01 JSON format, a service:

26. SHOULD return the OData JSON 4.01 format for requests with an `OData-MaxVersion` header value of `4.01`
27. MUST support the OData JSON 4.01 format in request payloads for requests with an `OData-Version` header value of `4.01`
28. MUST honor the `odata.etag` control information within `PUT`, `PATCH` or `DELETE` payloads, if specified
29. MUST support returning, in the final response to an asynchronous request, the `application/json` payload that would have been returned had the operation completed synchronously

Appendix A. References

This appendix contains the normative and informative references that are used in this document.

While any hyperlinks included in this appendix were valid at the time of publication, OASIS cannot guarantee their long-term validity.

A.1 Normative References

The following documents are referenced in such a way that some or all of their content constitutes requirements of this document.

[OData-ABNF]

ABNF components: OData ABNF Construction Rules Version 4.02 and OData ABNF Test Cases.
See link in "Related work" section on cover page.

[OData-CSDL]

OData Common Schema Definition Language (CSDL) JSON Representation Version 4.02.
See link in "Related work" section on cover page.

OData Common Schema Definition Language (CSDL) XML Representation Version 4.02.
See link in "Related work" section on cover page.

[OData-Protocol]

OData Version 4.02. Part 1: Protocol.
See link in "Related work" section on cover page.

[OData-URL]

OData Version 4.02. Part 2: URL Conventions.
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.

[RFC2119]

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997
https://www.rfc-editor.org/info/rfc2119.

[RFC3986]

Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", IETF RFC3986, January 2005 https://tools.ietf.org/html/rfc3986.

[RFC3987]

Duerst, M. and, M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, January 2005 https://tools.ietf.org/html/rfc3987.

[RFC4648]

Josefsson, S,, "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006 https://tools.ietf.org/html/rfc4648.

[RFC5646]

Phillips, A., Ed., and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, September 2009 http://tools.ietf.org/html/rfc5646.

[RFC7493]

Bray, T., Ed., "The I-JSON Message Format", RFC7493, March 2015 https://tools.ietf.org/html/rfc7493.

[RFC7946]

Howard Butler, Martin Daly, Alan Doyle, Sean Gillies, Stefan Hagen and Tim Schaub, "The GeoJSON Format", RFC 7946, August 2016. http://tools.ietf.org/html/rfc7946.

[RFC8174]

Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017
https://www.rfc-editor.org/info/rfc8174.

[RFC8259]

Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 8259, December 2017 http://tools.ietf.org/html/rfc8259.

A.2 Informative References

[ECMAScript]

ECMAScript 2023 Language Specification, 14th Edition, June 2023. Standard ECMA-262.
https://www.ecma-international.org/publications-and-standards/standards/ecma-262/.

Appendix B. Safety, Security and Privacy Considerations

This specification raises no security issues.

This section is provided as a service to the application developers, information providers, and users of OData version 4.0 giving some references to starting points for securing OData services as specified. OData is a REST-full multi-format service that depends on other services and thus inherits both sides of the coin, security enhancements and concerns alike from the latter.

For JSON-relevant security implications please cf. at least the relevant subsections of **[RFC8259]** as starting point.

Appendix C. Acknowledgments

C.1 Special Thanks

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

C.2 Participants

OData TC Members:

First Name	Last Name	Company
George	Ericson	Dell
Hubert	Heijkers	IBM
Ling	Jin	IBM
Stefan	Hagen	Individual
Michael	Pizzo	Microsoft
Christof	Sprenger	Microsoft
Ralf	Handl	SAP SE
Gerald	Krause	SAP SE
Heiko	Theißen	SAP SE
Mark	Biamonte	Progress Software
Martin	Zurmühl	SAP SE

Appendix D. Revision History

Revision	Date	Editor	Changes Made
Working Draft 01	2023-07-20	Ralf Handl	Import material from OData JSON Format Version 4.01

Appendix E. Notices

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

As stated in the OASIS IPR Policy, the following three paragraphs in brackets apply to OASIS Standards Final Deliverable documents (Committee Specification, Candidate OASIS Standard, OASIS Standard, or Approved Errata).

[OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Standards Final Deliverable, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this deliverable.]

[OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this OASIS Standards Final Deliverable by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this OASIS Standards Final Deliverable. OASIS may include such claims on its website, but disclaims any obligation to do so.]

[OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this OASIS Standards Final Deliverable or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by

an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Standards Final Deliverable, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.]