
OData JSON Format Version 4.02

Committee Specification Draft 01

28 February 2024

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. 28 February 2024. 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

Copyright © OASIS Open 2024. All Rights Reserved.

Distributed under the terms of the OASIS [IPR Policy](#).

The name “OASIS” is a trademark of [OASIS](#), the owner and developer of this specification, and should be used only to refer to the organization and its official outputs.

For complete copyright information please see the full Notices section in an Appendix [below](#).

Table of Contents

[1 Introduction](#)

[1.1 Changes from Earlier Versions](#)

[1.2 Glossary](#)

[1.2.1 Definitions of Terms](#)

[1.2.2 Acronyms and Abbreviations](#)

[1.2.3 Document Conventions](#)

[2 JSON Format Design](#)

[3 Requesting the JSON Format](#)

[3.1 Controlling the Amount of Control Information in Responses](#)

[3.1.1 metadata=minimal \(odata.metadata=minimal\)](#)

[3.1.2 metadata=full \(odata.metadata=full\)](#)

[3.1.3 metadata=none \(odata.metadata=none\)](#)

[3.2 Controlling the Representation of Numbers](#)

[4 Common Characteristics](#)

[4.1 Header Content-Type](#)

[4.2 Message Body](#)

[4.3 Relative URLs](#)

[4.4 Payload Ordering Constraints](#)

[4.5 Control Information](#)

[4.5.1 Control Information: context \(odata.context\)](#)

[4.5.2 Control Information: metadataEtag \(odata.metadataEtag\)](#)

[4.5.3 Control Information: type \(odata.type\)](#)

[4.5.4 Control Information: count \(odata.count\)](#)

[4.5.5 Control Information: nextLink \(odata.nextLink\)](#)

[4.5.6 Control Information: delta \(odata.delta\)](#)

[4.5.7 Control Information: deltaLink \(odata.deltaLink\)](#)

[4.5.8 Control Information: id \(odata.id\)](#)

[4.5.9 Control Information: editLink and readLink \(odata.editLink and odata.readLink\)](#)

[4.5.10 Control Information: etag \(odata.etag\)](#)

[4.5.11 Control Information: navigationLink and associationLink \(odata.navigationLink and odata.associationLink\)](#)

[4.5.12 Control Information: media* \(odata.media*\)](#)

[4.5.13 Control Information: removed \(odata.removed\)](#)

[4.5.14 Control Information: collectionAnnotations \(odata.collectionAnnotations\)](#)

[5 Service Document](#)

[6 Entity](#)

[7 Structural Property](#)

[7.1 Primitive Value](#)

[7.2 Complex Value](#)

[7.3 Collection of Primitive Values](#)

[7.4 Collection of Complex Values](#)

[7.5 Untyped Value](#)

[8 Navigation Property](#)

[8.1 Navigation Link](#)

[8.2 Association Link](#)

[8.3 Expanded Navigation Property](#)

[8.4 Deep Insert](#)

[8.5 Bind Operation](#)

[8.6 Collection ETag](#)

[9 Stream Property](#)

[10 Media Entity](#)

[11 Individual Property or Operation Response](#)

[12 Collection of Operation Responses](#)

[13 Collection of Entities](#)

[14 Entity Reference](#)

[15 Delta Payload](#)

[15.1 Delta Responses](#)

[15.2 Added/Changed Entity](#)

[15.3 Deleted Entity](#)

[15.4 Added Link](#)

[15.5 Deleted Link](#)

[15.6 Update a Collection of Entities](#)

[16 Bound Function](#)

[17 Bound Action](#)

[18 Action Invocation](#)

[19 Batch Requests and Responses](#)

[19.1 Batch Request](#)

[19.2 Referencing New Entities](#)

[19.3 Referencing an ETag](#)

[19.4 Referencing Response Body Values](#)

[19.5 Processing a Batch Request](#)

[19.6 Batch Response](#)

[19.7 Asynchronous Batch Requests](#)

[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](#)

[21 Error Handling](#)

[21.1 Error Response](#)

[21.2 In-Stream Error](#)

[21.3 Error Information in a Success Payload](#)

[21.3.1 Primitive Value Errors](#)

[21.3.2 Structured Type Errors](#)

[21.3.3 Collection Errors](#)

[22 Extensibility](#)

[23 Conformance](#)

[A References](#)

[A.1 Normative References](#)

[A.2 Informative References](#)

[B Safety, Security and Privacy Considerations](#)

[C Acknowledgments](#)

[C.1 Special Thanks](#)

[C.2 Participants](#)

[D Revision History](#)

[E Notices](#)

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

Section	Feature / Change	Issue
Section 4.5.12	mediaContentType can be null	ODATA-1470

1.2 Glossary

1.2.1 Definitions of Terms

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.

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+smart
-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 3.1.11.1 from <https://github.com/jgm/pandoc/releases/tag/3.1.11.1>.

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` 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` media 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` is requested. Otherwise it **MUST** be the first property of any JSON response that allows this control information (this excludes for example [error responses](#)).

The `context` control information **MUST** also be included in requests and responses for entities whose entity set cannot be determined from the context URL 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-version` 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`.

It **MAY** appear in other cases in requests and responses if its value does not contradict the type 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.

The `type` control information can be absent in properties nested in an instance of type `Edm.Untyped`. In particular, individual primitive values within a collection cannot have `type` control information.

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,
  ...
}
```

[4.5.4 Control Information: `count` \(`odata.count`\)](#)

The `count` control information occurs only in responses and can annotate any collection, see [\[OData-Protocol\]](#) section 11.2.5.5 System Query Option `$count`. Its value is an `Edm.Int64` value corresponding to the total count of members in the collection represented by the request.

[4.5.5 Control Information: `nextLink` \(`odata.nextLink`\)](#)

The `nextLink` control information indicates that a response is only a subset of the requested collection. It contains a URL that allows retrieving the next subset of the requested collection.

This control information can also be applied to [expanded to-many navigation properties](#).

[4.5.6 Control Information: `delta` \(`odata.delta`\)](#)

The `delta` control information is applied to a collection-valued navigation property within an [added/changed entity](#) in a delta payload to represent changes in membership or value of nested entities.

[4.5.7 Control Information: `deltaLink` \(`odata.deltaLink`\)](#)

The `deltaLink` control information contains a URL that can be used to retrieve changes to the current set of results. The `deltaLink` control information MUST only appear on the last page of results. A page of results MUST NOT have both a `deltaLink` control information and a `nextLink` control information.

[4.5.8 Control Information: `id` \(`odata.id`\)](#)

The `id` control information contains the entity-id, see [\[OData-Protocol\]](#). By convention the entity-id is identical to the canonical URL of the entity, as defined in [\[OData-URL\]](#).

The `id` control information MUST appear in responses if [metadata=full](#) is requested, or if [metadata=minimal](#) is requested and any of a non-transient entity's key fields are omitted from the response *or* the entity-id is not identical to the canonical URL of the entity after

- IRI-to-URI conversion as defined in [\[RFC3987\]](#),
- relative resolution as defined in section 5.2 of [\[RFC3986\]](#), and
- percent-encoding normalization as defined in section 6 of [\[RFC3986\]](#).

Note that the entity-id MUST be invariant across languages, so if key values are language dependent then the `id` MUST be included if it does not match convention for the localized key values. If the `id` is represented, it MAY be a [relative URL](#).

If the entity is transient (i.e. cannot be read or updated), the `id` control information MUST appear in OData 4.0 payloads and have the `null` value. In 4.01 payloads transient entities need not have the `id` control information, and 4.01 clients MUST treat entities with neither `id` control information nor a full set of key properties as transient entities.

The `id` control information MUST NOT appear for a collection. Its meaning in this context is reserved for future versions of this specification.

Entities with `id` equal to `null` cannot be compared to other entities, reread, or updated. If [metadata=minimal](#) is specified and the `id` is not present in the entity, then the canonical URL MUST be used as the entity-id.

[4.5.9 Control Information: editLink and readLink \(odata.editLink and odata.readLink\)](#)

The `editLink` control information contains the edit URL of the entity; see [\[OData-Protocol\]](#).

The `readLink` control information contains the read URL of the entity or collection; see [\[OData-Protocol\]](#).

The `editLink` and `readLink` control information is ignored in request payloads and not written in responses if [metadata=none](#) is requested.

The default value of both the edit URL and read URL is the entity's [entity-id](#) appended with a cast segment to the type of the entity if its type is derived from the declared type of the entity set. If neither the `editLink` nor the `readLink` control information is present in an entity, the client uses this default value for the edit URL.

For updatable entities:

- The `editLink` control information is written if [metadata=full](#) is requested or if [metadata=minimal](#) is requested and the edit URL differs from the default value of the edit URL.
- The `readLink` control information is written if the read URL is different from the edit URL. If no `readLink` control information is present, the read URL is identical to the edit URL.

For read-only entities:

- The `readLink` control information is written if [metadata=full](#) is requested or if [metadata=minimal](#) is requested and its value differs from the default value of the read URL.
- The `readLink` control information may also be written if [metadata=minimal](#) is specified in order to signal that an individual entity is read-only.

For collections:

- The `readLink` control information, if written, MUST be the request URL that produced the collection.
- The `editLink` control information MUST NOT be written as its meaning in this context is reserved for future versions of this specification.

[4.5.10 Control Information: etag \(odata.etag\)](#)

The `etag` control information MAY be applied to an [entity](#) or collection in a response. The value of the control information is an entity tag (ETag) which is an opaque string value that can be used in a subsequent request to determine if the value of the entity or collection has changed.

For details on how ETags are used, see [\[OData-Protocol\]](#).

The `etag` control information is ignored in request payloads for single entities and not written in responses if [metadata=none](#) is requested.

[4.5.11 Control Information: navigationLink and associationLink \(odata.navigationLink and odata.associationLink\)](#)

The `navigationLink` control information in a response contains a *navigation URL* that can be used to retrieve an entity or collection of entities related to the current entity via a [navigation property](#).

The *default computed value of a navigation URL* is the value of the [read URL](#) appended with a segment containing the name of the navigation property. The service MAY omit the `navigationLink` control information if

[metadata=minimal](#) has been specified on the request and the navigation link matches this computed value.

The `associationLink` control information in a response contains an *association URL* that can be used to retrieve a reference to an entity or a collection of references to entities related to the current entity via a navigation property.

The *default computed value of an association URL* is the value of the navigation URL appended with `/$ref`. The service MAY omit the `associationLink` control information if the association link matches this computed value.

The `navigationLink` and `associationLink` control information is ignored in request payloads and not written in responses if [metadata=none](#) is requested.

[4.5.12 Control Information: media* \(odata.media*\)](#)

For [media entities](#) and [stream properties](#) at least one of the control information `mediaEditLink` and `mediaReadLink` MUST be included in responses if they don't follow standard URL conventions as defined in [\[OData-URL\]](#), sections 4.6 Addressing a property and 4.14 Addressing the Media Stream of a Media Entity, or if [metadata=full](#) is requested.

The `mediaEditLink` control information contains a URL that can be used to update the binary stream associated with the media entity or stream property. It MUST be included for updatable streams if it differs from standard URL conventions relative to the edit link of the entity.

The `mediaReadLink` control information contains a URL that can be used to read the binary stream associated with the media entity or stream property. It MUST be included if its value differs from the value of the associated `mediaEditLink`, if present, or if it doesn't follow standard URL conventions relative to the read link of the entity and the associated `mediaEditLink` is not present.

The `mediaContentType` control information MAY be included; its value SHOULD match the media type of the binary stream represented by the `mediaReadLink` URL. This is only a hint; the actual media type will be included in the `Content-Type` header when the resource is requested. The presence of `mediaContentType` with value `null` MAY be used to indicate the absence of a binary stream.

The `mediaEtag` control information MAY be included; its value is the ETag of the binary stream represented by this media entity or stream property.

The `media*` control information is not written in responses if [metadata=none](#) is requested.

If a stream property is provided inline in a request, the `mediaContentType` control information may be specified.

If a stream property is annotated with `Capabilities.MediaLocationUpdateSupported` (see [\[OData-VocCap\]](#)) and a value of `true`, clients MAY specify the `mediaEditLink` and/or `mediaReadLink` control information for that stream property in order to change the association between the stream property and a media stream.

In all other cases `media*` control information is ignored in request payloads.

Example 7:

```
{
  "@context": "http://host/service/$metadata#Employees/$entity",
  "@mediaReadLink": "Employees(1)/$value",
  "@mediaContentType": "image/jpeg",
  "ID": 1,
  ...
}
```

[4.5.13 Control Information: removed \(odata.removed\)](#)

The `removed` control information is used in [delta payloads](#) and indicates that the represented entity is (to be) deleted.

4.5.14 Control Information: collectionAnnotations (odata.collectionAnnotations)

The `collectionAnnotations` control information can be applied to a collection containing primitive members in order to annotate such primitive members. The value of the `collectionAnnotations` control information is an array of JSON objects containing an integer property `index`, specifying the zero-based ordinal index of the primitive item within the collection, along with any annotations that are to be applied to that primitive collection member.

Example 8: Annotating primitive values within a collection

```
{
  "@context": "http://host/service/$metadata#Employees/$entity",
  "ID": 1,
  "EmailAddresses@collectionAnnotations": [
    {
      "index": 0,
      "@OfficeCommunication.emailType": "Personal"
    },
    {
      "index": 2,
      "@OfficeCommunication.emailType": "Work"
    }
  ],
  "EmailAddresses": [
    "Julie@Swansworth.com",
    "JulieSwa@live.com",
    "Julie.Swansworth@work.com"
  ],
  ...
}
```

5 Service Document

A service document in JSON is represented as a single JSON object with at least the [context](#) control information and a property value.

The value of the [context](#) control information MUST be the URL of the metadata document, without any fragment part.

The value of the `value` property MUST be a JSON array containing one element for each entity set and function import with an explicit or default value of `true` for the attribute `IncludeInServiceDocument` and each singleton exposed by the service, see [\[OData-CSDLJSON\]](#) or [\[OData-CSDLXML\]](#).

Each element MUST be a JSON object with at least two name/value pairs, one with name `name` containing the name of the entity set, function import, or singleton, and one with name `url` containing the URL of the entity set, which may be an absolute or a [relative URL](#). It MAY contain a name/value pair with name `title` containing a human-readable, language-dependent title for the object.

JSON objects representing an entity set MAY contain an additional name/value pair with name `kind` and a value of `EntitySet`. If the `kind` name/value pair is not present, the object MUST represent an entity set.

JSON objects representing a function import MUST contain the `kind` name/value pair with a value of `FunctionImport`.

JSON objects representing a singleton MUST contain the `kind` name/value pair with a value of `Singleton`.

JSON objects representing a related service document MUST contain the `kind` name/value pair with a value of `ServiceDocument`.

Clients that encounter unknown values of the `kind` name/value pair not defined in this version of the specification MUST NOT stop processing and MUST NOT signal an error.

Service documents MAY contain [annotations](#) in any of its JSON objects. Services MUST NOT produce name/value pairs other than the ones explicitly defined in this section, and clients MUST ignore unknown name/value pairs.

Example 9:

```
{
  "@context": "http://host/service/$metadata",
  "value": [
    {
      "name": "Orders",
      "kind": "EntitySet",
      "url": "Orders"
    },
    {
      "name": "OrderItems",
      "title": "Order Details",
      "url": "OrderItems"
    },
    {
      "name": "TopProducts",
      "title": "Best-Selling Products",
      "kind": "FunctionImport",
      "url": "TopProducts"
    },
    {
      "name": "MainSupplier",
      "title": "Main Supplier",

```

```
    "kind": "Singleton",  
    "url": "MainSupplier"  
  },  
  {  
    "name": "Human Resources",  
    "kind": "ServiceDocument",  
    "url": "http://host/HR/"  
  }  
]  
}
```

6 Entity

An entity is serialized as a JSON object. It MAY contain [context](#), [type](#), or [deltaLink](#) control information.

Each [property](#) to be transmitted is represented as a name/value pair within the object. The order properties appear within the object is considered insignificant.

An entity in a payload may be a complete entity, a projected entity (see *System Query Option \$select* in [\[OData-Protocol\]](#)), or a partial entity update (see *Update an Entity* in [\[OData-Protocol\]](#)).

An entity representation can be (modified and) round-tripped to the service directly. The [context URL](#) is used in requests only as a base for [relative URLs](#).

Example 10: entity with metadata=minimal

```
{
  "@context": "http://host/service/$metadata#Customers/$entity",
  "ID": "ALFKI",
  "CompanyName": "Alfreds Futterkiste",
  "ContactName": "Maria Anders",
  "ContactTitle": "Sales Representative",
  "Phone": "030-0074321",
  "Fax": "030-0076545",
  "Address": {
    "Street": "Obere Str. 57",
    "City": "Berlin",
    "Region": null,
    "PostalCode": "D-12209"
  }
}
```

Example 11: entity with metadata=full

```
{
  "@context": "http://host/service/$metadata#Customers/$entity",
  "@id": "Customers('ALFKI')",
  "@etag": "W/\"MjAxMy0wNS0yN1QxMT01Ofo=\"",
  "@editLink": "Customers('ALFKI')",
  "ID": "ALFKI",
  "CompanyName": "Alfreds Futterkiste",
  "ContactName": "Maria Anders",
  "ContactTitle": "Sales Representative",
  "Phone": "030-0074321",
  "Fax": "030-0076545",
  "Address": {
    "Street": "Obere Str. 57",
    "City": "Berlin",
    "Region": null,
    "PostalCode": "D-12209",
    "Country@associationLink": "Customers('ALFKI')/Address/Country/$ref",
    "Country@navigationLink": "Customers('ALFKI')/Address/Country"
  },
  "Orders@associationLink": "Customers('ALFKI')/Orders/$ref",
  "Orders@navigationLink": "Customers('ALFKI')/Orders"
}
```

7 Structural Property

A property within an entity or complex type instance is represented as a name/value pair. The name **MUST** be the name of the property; a non-null value is represented depending on its type as a [primitive value](#), a [complex value](#), a [collection of primitive values](#), or a [collection of complex values](#).

Null values are represented as the JSON literal `null`.

7.1 Primitive Value

Primitive values are represented following the rules of [\[RFC8259\]](#).

Values of type `Edm.Boolean` are represented as the JSON literals `true` and `false`

Values of types `Edm.Byte`, `Edm.SByte`, `Edm.Int16`, `Edm.Int32`, `Edm.Int64`, `Edm.Single`, `Edm.Double`, and `Edm.Decimal` are represented as JSON numbers, except for `-INF`, `INF`, and `NaN` which are represented as strings.

Values of type `Edm.String` are represented as JSON strings, using the JSON string escaping rules.

Values of type `Edm.Binary`, `Edm.Date`, `Edm.DateTimeOffset`, `Edm.Duration`, `Edm.Guid`, and `Edm.TimeOfDay` are represented as JSON strings whose content satisfies the rules `binaryValue`, `dateValue`, `dateTimeOffsetValue`, `durationValue`, `guidValue`, and `timeOfDayValue` respectively, in [\[OData-ABNF\]](#).

Primitive values that cannot be represented, for example due to server conversion issues or IEEE754 limitations on the size of an `Edm.Int64` or `Edm.Decimal` value, are annotated with the `Core.ValueException` term. In this case, the payload **MAY** include an approximation of the value and **MAY** specify a string representation of the exact value in the `value` property of the annotation.

Enumeration values are represented as JSON strings whose content satisfies the rule `enumValue` in [\[OData-ABNF\]](#). The preferred representation is the `enumerationMember`. If no `enumerationMember` (or combination of named enumeration members) is available, the `enumMemberValue` representation may be used.

Geography and geometry values are represented as geometry types as defined in [\[RFC7946\]](#).

Geography and geometry types have the same representation in a JSON payload. Whether the value represents a geography type or geometry type is inferred from its usage or specified using the [type](#) control information.

Example 12:

```
{
  "NullValue": null,
  "TrueValue": true,
  "FalseValue": false,
  "BinaryValue": "T0RhGE",
  "IntegerValue": -128,
  "DoubleValue": 3.1415926535897931,
  "SingleValue": "INF",
  "DecimalValue": 34.95,
  "StringValue": "Say \"Hello\", \nthen go",
  "DateValue": "2012-12-03",
  "DateTimeOffsetValue": "2012-12-03T07:16:23Z",
  "DurationValue": "P12DT23H59M59.999999999999S",
  "TimeOfDayValue": "07:59:59.999",
  "GuidValue": "01234567-89ab-cdef-0123-456789abcdef",
  "Int64Value": 0,
  "ColorEnumValue": "Yellow",
  "GeographyPoint": {"type": "Point", "coordinates": [142.1, 64.1]}
}
```


7.2 Complex Value

A complex value is represented as a single JSON object containing one name/value pair for each property that makes up the complex type. Each property value is formatted as appropriate for the type of the property.

It MAY have name/value pairs for [instance annotations](#) and control information.

Example 13:

```
{
  "@context": "http://host/service/$metadata#Customers/$entity",
  ...
  "Address": {
    "Street": "Obere Str. 57",
    "City": "Berlin",
    "Region": null,
    "PostalCode": "D-12209"
  }
}
```

A complex value with no selected properties, or no defined properties (such as an empty open complex type or complex type with no structural properties) is represented as an empty JSON object.

7.3 Collection of Primitive Values

A collection of primitive values is represented as a JSON array; each element in the array is the representation of a [primitive value](#). A JSON literal `null` represents a null value within the collection. An empty collection is represented as an empty array.

Example 14: partial collection of strings with next link

```
{
  "@context": "http://host/service/$metadata#Customers/$entity",
  ...
  "EmailAddresses": [
    "Julie@Swansworth.com",
    "Julie.Swansworth@work.com"
  ],
  "EmailAddresses@nextLink": "..."
}
```

A collection of primitive values that occurs in a property of type `Edm.Untyped` is interpreted as a collection of `Edm.Boolean`, `Edm.String`, and `Edm.Decimal` values, depending on the JavaScript type.

7.4 Collection of Complex Values

A collection of complex values is represented as a JSON array; each element in the array is the representation of a [complex value](#). A JSON literal `null` represents a null value within the collection. An empty collection is represented as an empty array.

Example 15: partial collection of complex values with next link

```
{
  "PhoneNumbers": [
    {
      "Number": "425-555-1212",
      "Type": "Home"
    },
    {

```

```
    "@type": "#Model.CellPhoneNumber",  
    "Number": "425-555-0178",  
    "Type": "Cell",  
    "Carrier": "Sprint"  
  }  
],  
  "PhoneNumbers@nextLink": "..."  
}
```

7.5 Untyped Value

OData 4.01 adds the built-in abstract types `Edm.Untyped` and `Collection(Edm.Untyped)` that services can use to advertise in metadata that there is a property of a particular name present, but there is no type to describe the structure of the property's values.

The value of an `Edm.Untyped` property MAY be a primitive value, a structural value, or a collection. If a collection, it may contain any combination of primitive values, structural values, and collections.

The value of a property of type `Collection(Edm.Untyped)` MUST be a collection, and it MAY contain any combination of primitive values, structural values, and collections.

Untyped values are the only place where a collection can directly contain a collection, or a collection can contain a mix of primitive values, structural values, and collections.

All children of an untyped property are assumed to be untyped unless they are annotated with the [type](#) control information, in which case they MUST conform to the type described by the control information.

8 Navigation Property

A navigation property is a reference from a source entity to zero or more related entities.

8.1 Navigation Link

The navigation link for a navigation property is represented as a [navigationLink](#) control information on the navigation property. Its value is an absolute or [relative URL](#) that allows retrieving the related entity or collection of entities.

The navigation link for a navigation property is only represented if the client requests `metadata=full` or the navigation link cannot be computed, e.g. if it is within a collection of complex type instances. If it is represented it **MUST** immediately precede the expanded navigation property if the latter is represented.

Example 16:

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

8.2 Association Link

The association link for a navigation property is represented as an [associationLink](#) control information on the navigation property. Its value is an absolute or [relative URL](#) that can be used to retrieve the reference or collection of references to the related entity or entities.

The association link for a navigation property is only represented if the client requests `metadata=full` or the association link cannot be computed by appending `/$ref` to the navigation link. If it is represented, it **MUST** immediately precede the navigation link if the latter is represented, otherwise it **MUST** immediately precede the expanded navigation property if it is represented.

Example 17:

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

8.3 Expanded Navigation Property

An expanded navigation property is represented as a name/value pair where the name is the name of the navigation property, and the value is the representation of the related entity or collection of entities.

If at most one entity can be related, the value is the representation of the related entity, or `null` if no entity is currently related.

If a collection of entities can be related, it is represented as a JSON array. Each element is the [representation of an entity](#) or the [representation of an entity reference](#). An empty collection of entities (one that contains no entities) is represented as an empty JSON array. The navigation property **MAY** include [context](#), [type](#), [count](#), or [nextLink](#) control information. If a navigation property is expanded with the suffix `/$count`, only the [count](#) control information is represented.

Example 18:

```
{
  "@context": "http://host/service/$metadata#Customers/$entity",
  "Orders@count": 42,
  "Orders": [ ... ],
  "Orders@nextLink": "...",
  ...
}
```

8.4 Deep Insert

When inserting a new entity with a POST request, related new entities MAY be specified using the same representation as for an [expanded navigation property](#).

Deep inserts are not allowed in update operations using PUT or PATCH requests.

Example 19: inserting a new order for a new customer with order items related to existing products:

```
{
  "ID": 11643,
  "Amount": 100,
  ...,
  "Customer": {
    "ID": "ANEWONE",
    ...
  },
  "Items": [
    {
      "Product": { "@id": "Products(28)" },
      "Quantity": 1,
      ...
    },
    {
      "Product": { "@id": "Products(39)" },
      "Quantity": 5,
      ...
    }
  ]
}
```

8.5 Bind Operation

When inserting or updating an entity, relationships of navigation properties MAY be inserted or updated via bind operations.

For requests containing an `odata-version` header with a value of 4.0, a bind operation is encoded as a property control information `odata.bind` on the navigation property it belongs to and has a single value for single-valued navigation properties or an array of values for collection navigation properties. For nullable single-valued navigation properties the value `null` may be used to remove the relationship.

Example 20: assign an existing product to an existing category with a partial update request against the product

```
PATCH http://host/service/Products(42) HTTP/1.1
Content-Type: application/json

{
  "Category@odata.bind": "Categories(6)"
}
```

The values are the [ids](#) of the related entities. They MAY be absolute or [relative URLs](#).

For requests containing an `OData-Version` header with a value of 4.01, a relationship is bound to an existing entity using the same representation as for an [expanded entity reference](#).

Example 21: assign an existing product to an existing category with a partial update request against the product

```
PATCH http://host/service/Products(42) HTTP/1.1
Content-Type: application/json

{
  "Category": {"@id": "Categories(6)"}
}
```

Example 22: submit a partial update request to:

- *modify the name of an existing category*
- *assign an existing product with the id 42 to the category*
- *assign an existing product 57 to the category and update its name*
- *create a new product named `Wedges` and assign it to the category*

At the end of the request, the updated category contains exactly the three specified products.

```
PATCH http://host/service/Categories(6) HTTP/1.1
Content-Type: application/json

{
  "Name": "UpdatedCategory",
  "Products": [
    {
      "@id": "Products(42)"
    },
    {
      "@id": "Products(57)",
      "Name": "Widgets"
    },
    {
      "Name": "Wedges"
    }
  ]
}
```

OData 4.01 services MUST support both the OData 4.0 representation, for requests containing an `OData-Version` header with a value of 4.0, and the OData 4.01 representation, for requests containing an `OData-Version` header with a value of 4.01. Clients MUST NOT use `@odata.bind` in requests with an `OData-Version` header with a value of 4.01.

For insert operations collection navigation property bind operations and deep insert operations can be combined. For OData 4.0 requests, the bind operations MUST appear before the deep insert operations in the payload.

For update operations a bind operation on a collection navigation property adds additional relationships, it does not replace existing relationships, while bind operations on an entity navigation property update the relationship.

8.6 Collection ETag

The ETag for a collection of related entities is represented as [etag](#) control information on the navigation property. Its value is an opaque string that can be used in a subsequent request to determine if the collection has changed.

Services MAY include this control information as appropriate.

Example 23: ETag for a collection of related entities

```
{
  "@context": "http://host/service/$metadata#Orders/$entity",
  "@id": "Orders(1234)",
  "@etag": "W/\"MjAxMy0wNS0yN1QxMT01OFo=\"",
  "ID": 1234,
  "Items@etag": "W/\"MjAxOS0wMy0xM1QxMDoyM1o=\"",
  ...
}
```

Note: the collection ETag for a navigation property may or may not be identical to the ETag of the containing entity, the example shows a different ETag for the `Items` collection.

9 Stream Property

An entity or complex type instance can have one or more stream properties.

The actual stream data is not usually contained in the representation. Instead stream property data is generally read and edited via URLs.

- Stream properties requested with `$select` or included in the default selection are represented by [media*](#) control information.
- Stream properties requested with `$expand` or implicitly expanded are represented as a property with its value.

See [\[OData-Protocol\]](#) for details on the system query options `$select` and `$expand`.

Depending on the [metadata level](#), the stream property MAY be annotated to provide the read link, edit link, media type, and ETag of the media stream through their `media*` control information.

If the actual stream data is included inline, the control information [mediaContentType](#) MUST be present to indicate how the included stream property value is represented. Stream property values of media type `application/json` or one of its subtypes, optionally with format parameters, are represented as native JSON. Values of top-level type `text` with an explicit or default `charset` of `utf-8` or `us-ascii`, for example `text/plain`, are represented as a string, with JSON string escaping rules applied. Included stream data of other media types is represented as a `base64url`-encoded string value, see [\[RFC4648\]](#), section 5.

If the included stream property has no value, the non-existing stream data is represented as `null` and the control information [mediaContentType](#) is not necessary.

Example 24:

```
{
  "@context": "http://host/service/$metadata#Products/$entity",
  ...
  "Thumbnail@mediaReadLink": "http://server/Thumbnail1546.jpg",
  "Thumbnail@mediaEditLink": "http://server/uploads/Thumbnail1546.jpg",
  "Thumbnail@mediaContentType": "image/jpeg",
  "Thumbnail@mediaEtag": "W/\"####\"",
  "Thumbnail": "...base64url encoded value...",
  ...
}
```

10 Media Entity

Media entities are entities that describe a media resource, for example a photo. They are represented as entities that contain additional [media*](#) control information.

If the actual stream data for the media entity is included, it is represented as property named `$value` whose string value is the base64url-encoded value of the media stream, see [RFC4648](#).

Example 25:

```
{
  "@context": "http://host/service/$metadata#Employees/$entity",
  "@mediaReadLink": "Employees(1)/$value",
  "@mediaContentType": "image/jpeg",
  "$value": "...base64url encoded value...",
  "ID": 1,
  ...
}
```


11 Individual Property or Operation Response

An individual property or operation response is represented as a JSON object.

A single-valued property or operation response that has the `null` value does not have a representation; see [\[OData-Protocol\]](#).

A property or operation response that is of a primitive type is represented as an object with a single name/value pair, whose name is `value` and whose value is a [primitive value](#).

A property or operation response that is of complex type is represented as a [complex value](#).

A property or operation response that is of a collection type is represented as an object with a single name/value pair whose name is `value`. Its value is the JSON representation of a [collection of complex type values](#) or [collection of primitive values](#).

Example 26: primitive value

```
{
  "@context": "http://host/service/$metadata#Edm.String",
  "value": "Pilar Ackerman"
}
```

Example 27: collection of primitive values

```
{
  "@context": "http://host/service/$metadata#Collection(Edm.String)",
  "value": ["small", "medium", "extra large"]
}
```

Example 28: empty collection of primitive values

```
{
  "@context": "http://host/service/$metadata#Collection(Edm.String)",
  "value": []
}
```

Example 29: complex value

```
{
  "@context": "http://host/service/$metadata#Model.Address",
  "Street": "12345 Grant Street",
  "City": "Taft",
  "Region": "Ohio",
  "PostalCode": "OH 98052",
  "Country@navigationLink": "Countries('US')"
}
```

Example 30: empty collection of complex values

```
{
  "@context": "http://host/service/$metadata#Collection(Model.Address)",
  "value": []
}
```

Note: the context URL is optional in requests.

12 Collection of Operation Responses

Invoking a bound action or function with `/each` on each member of a collection in one request results in a collection of operation results, which is represented as a JSON object containing a name/value pair named `value`. It MAY contain `context`, `type`, `count`, or `nextLink` control information.

If present, the `context` control information MUST be the first name/value pair in the response.

The `count` name/value pair represents the number of operation responses in the collection. If present and the `streaming=true` media type parameter is set, it MUST come before the `value` name/value pair. If the response represents a partial result, the `count` name/value pair MUST appear in the first partial response, and it MAY appear in subsequent partial responses (in which case it may vary from response to response).

The value of the `value` name/value pair is an array of objects, each object representing a single [operation response](#). Note: if the operation response is a collection, each single operation response object itself contains a name/value pair named `value`.

13 Collection of Entities

A collection of entities is represented as a JSON object containing a name/value pair named `value`. It MAY contain [context](#), [type](#), [count](#), [nextLink](#), or [deltaLink](#) control information.

If present, the `context` control information MUST be the first name/value pair in the response.

The `count` name/value pair represents the number of entities in the collection. If present and the [streaming=true](#) media type parameter is set, it MUST come before the `value` name/value pair. If the response represents a partial result, the `count` name/value pair MUST appear in the first partial response, and it MAY appear in subsequent partial responses (in which case it may vary from response to response).

The value of the `value` name/value pair is a JSON array where each element is [representation of an entity](#) or a [representation of an entity reference](#). An empty collection is represented as an empty JSON array.

Functions or actions that are bound to this collection of entities are advertised in the “wrapper object” in the same way as [functions](#) or [actions](#) are advertised in the object representing a single entity.

The [nextLink](#) control information MUST be included in a response that represents a partial result.

Example 28:

```
{
  "@context": "...",
  "@count": 37,
  "value": [
    { ... },
    { ... },
    { ... }
  ],
  "@nextLink": "...?$skiptoken=342r89"
}
```

14 Entity Reference

An entity reference (see [\[OData-Protocol\]](#)) MAY take the place of an entity in a JSON payload, based on the client request. It is serialized as a JSON object that MUST contain the [id](#) of the referenced entity and MAY contain the [type](#) control information and [instance annotations](#), but no additional properties or control information.

A collection of entity references is represented as a [collection of entities](#), with entity reference representations instead of entity representations as items in the array value of the `value` name/value pair.

The outermost JSON object in a response MUST contain a [context](#) control information and MAY contain [count](#), [nextLink](#), or [deltaLink](#) control information.

Example 31: entity reference to order 10643

```
{
  "@context": "http://host/service/$metadata#$ref",
  "@id": "Orders(10643)"
}
```

Example 32: collection of entity references

```
{
  "@context": "http://host/service/$metadata#Collection($ref)",
  "value": [
    { "@id": "Orders(10643)" },
    { "@id": "Orders(10759)" }
  ]
}
```

15 Delta Payload

The non-format specific aspects of the delta handling are described in the section “Requesting Changes” in [\[OData-Protocol\]](#).

15.1 Delta Responses

Responses from a delta request are returned as a JSON object.

The JSON object for a delta response to a single entity is either an [added](#), [changed](#), or [deleted entity](#).

The JSON object for a delta response to a collection of entities MUST contain an array-valued property named `value` containing all [added](#), [changed](#), or [deleted entities](#), as well as [added links](#) or [deleted links](#) between entities, and MAY contain additional, unchanged entities.

If the delta response contains a partial list of changes, it MUST include a [next link](#) for the client to retrieve the next set of changes.

The last page of a delta response SHOULD contain a [delta link](#) in place of the [next link](#) for retrieving subsequent changes once the current set of changes has been applied to the initial set.

If an OData 4.01 delta response includes an expanded collection-valued navigation property inline (see [next section](#)), the expanded collection can be a partial list, in which case the expanded navigation property MUST have the [nextLink](#) control information applied to it. Following this chain of next links does not result in a delta link on the last page of the expanded collection.

If the response from the delta link contains a `count` control information, the returned number MUST include all added, changed, or deleted entities to be returned, as well as added or deleted links.

Example 33: a 4.01 delta response with five changes, in order of occurrence

1. ContactName for customer BOTTM was changed to Susan Halvenstern
2. Order 10643 was removed from customer ALFKI
3. Order 10645 was added to customer BOTTM
4. The shipping information for order 10643 was updated
5. Customer ANTON was deleted

```
{
  "@context": "http://host/service/$metadata#Customers/$delta",
  "@count": 5,
  "value": [
    {
      "@id": "Customers('BOTTM')",
      "ContactName": "Susan Halvenstern"
    },
    {
      "@context": "#Customers/$deletedLink",
      "source": "Customers('ALFKI')",
      "relationship": "Orders",
      "target": "Orders(10643)"
    },
    {
      "@context": "#Customers/$link",
      "source": "Customers('BOTTM')",
      "relationship": "Orders",
      "target": "Orders(10645)"
    }
  ],
  {
```

```

    "@context": "#Orders/$entity",
    "@id": "Orders(10643)",
    "ShippingAddress": {
      "Street": "23 Tsawassen Blvd.",
      "City": "Tsawassen",
      "Region": "BC",
      "PostalCode": "T2F 8M4"
    },
  },
  {
    "@context": "#Customers/$deletedEntity",
    "@removed": {
      "reason": "deleted"
    },
    "@id": "Customers('ANTON')"
  }
],
"@deltaLink": "Customers?$expand=Orders&$deltatoken=8015"
}

```

15.2 Added/Changed Entity

Added or changed entities within a delta response are represented as [entities](#).

Added entities MUST include all available selected properties and MAY include additional, unselected properties. Collection-valued properties are treated as atomic values; any collection-valued properties returned from a delta request MUST contain all current values for that collection.

Changed entities MUST include all available selected properties that have changed, and MAY include additional properties.

If a property of an entity is dependent upon the property of another entity within the expanded set of entities being tracked, then both the change to the dependent property as well as the change to the principal property or [added/deleted link](#) corresponding to the change to the dependent property are returned in the delta response.

Entities that are not part of the entity set specified by the context URL MUST include the [context](#) control information to specify the entity set of the entity, regardless of the specified [metadata](#) value.

Entities include control information for selected navigation links based on [metadata](#).

OData 4.0 payloads MUST NOT include expanded navigation properties inline; all changes MUST be represented as a flat array of added, deleted, or changed entities, along with added or deleted links.

OData 4.01 delta payloads MAY include expanded navigation properties inline. Related single entities are represented as either an [added/changed](#) entity, an [entity reference](#), a [deleted entity](#), or a null value (if no entity is related as the outcome of the change). Collection-valued navigation properties are represented either as a delta representation or as a full representation of the collection.

If the expanded navigation property represents a delta, it MUST be represented as an array-valued control information [delta](#) on the navigation property. [Added/changed](#) entities or [entity references](#) are added to the collection. [Deleted entities](#) MAY be specified in a nested delta representation to represent entities no longer part of the collection. If the deleted entity specifies a `reason` as `deleted`, then the entity is both removed from the collection and deleted, otherwise it is removed from the collection and only deleted if the navigation property is a containment navigation property. The array MUST NOT contain [added](#) or [deleted links](#).

Example 34: 4.01 delta response customers with expanded orders represented inline as a delta

1. Customer BOTTOM:

1. ContactName was changed to Susan Halvenstern

2. Order 10645 was added
2. Customer ALFKI:
 1. Order 10643 was removed
3. Customer ANTON was deleted

```
{
  "@context": "http://host/service/$metadata#Customers/$delta",
  "@count": 3,
  "value": [
    {
      "@id": "Customers('BOTTM')",
      "ContactName": "Susan Halvenstern",
      "Orders@delta": [
        {
          "@id": "Orders(10645)"
        }
      ]
    },
    {
      "@id": "Customers('ALFKI')",
      "Orders@delta": [
        {
          "@context": "#Orders/$deletedEntity",
          "@removed": {
            "reason": "changed"
          },
          "@id": "Orders(10643)"
        }
      ]
    },
    {
      "@context": "#Customers/$deletedEntity",
      "@removed": {
        "reason": "deleted"
      },
      "@id": "Customers('ANTON')"
    }
  ],
  "@deltaLink": "Customers?$expand=Orders&$deltatoken=8015"
}
```

If the expanded navigation property is a full representation of the collection, it MUST be represented as an expanded navigation property, and its array value represents the full set of entities related according to that relationship and satisfying any specified expand options. Members of the array MUST be represented as [added/changed](#) entities or [entity references](#) and MUST NOT include added links, deleted links, or deleted entities. Any entity not represented in the collection has either been removed, deleted, or changed such that it no longer satisfies the expand options in the defining query. In any case, clients SHOULD NOT receive additional notifications for such removed entities.

Example 35: 4.01 delta response for a single entity with an expanded navigation property containing only a partial list of related entities (as indicated with a [next link](#))

```
{
  "@context": "http://host/service/$metadata#Customers/$entity/$delta",
  ...
  "Orders@count": 42,
  "Orders": [ ... ],
  "Orders@nextLink": "...",
  ...
}
```

```
"@deltaLink": "Customers('ALFKI')?$expand=Orders&$deltatoken=9711"
}
```

15.3 Deleted Entity

Deleted entities in JSON are returned as deleted-entity objects. Delta responses MUST contain a deleted-entity object for each deleted entity, including deleted expanded entities that are not related through a containment navigation property. The service MAY additionally include expanded entities related through a containment navigation property in which case it MUST include those in any returned count of enumerated changes.

The representation of deleted-entity objects differs between OData 4.0 and OData 4.01.

In OData 4.0 payloads the deleted-entity object MUST include the following properties, regardless of the specified [metadata](#) value:

- Control information [context](#) — The context URL fragment MUST be `#{entity-set}/$deletedEntity`, where `{entity-set}` is the entity set of the deleted entity
- `id` — The [id](#) of the deleted entity (same as the [id](#) returned or computed when calling GET on resource), which may be absolute or [relative](#)

In OData 4.0 payloads the deleted-entity object MAY include the following optional property, regardless of the specified [metadata](#) value, and MAY include [annotations](#):

- `reason` — either `deleted`, if the entity was deleted (destroyed), or `changed` if the entity was removed from membership in the result (i.e., due to a data change).

Example 36: deleted entity in OData 4.0 response — note that `id` is a property, not control information

```
{
  "@context": "#Customers/$deletedEntity",
  "reason": "deleted",
  "id": "Customers('ANTON')"
}
```

In OData 4.01 payloads the deleted-entity object MUST include the following properties, regardless of the specified [metadata](#) value:

- Control information [removed](#), whose value is an object that MAY contain a property named `reason`. If present, the value of `reason` MUST be either `deleted` if the entity was deleted (destroyed), or `changed` if the entity was removed from membership in the result either due to change in value such that the entity no longer matches the defining query or because the entity was removed from the collection. The object MAY include [annotations](#), and clients SHOULD NOT error due to the presence of additional properties that MAY be defined by future versions of this specification. For [ordered payloads](#), the control information [removed](#) MUST immediately follow the [context](#) control information, if present, otherwise it MUST be the first property in the deleted entity.
- Control information [id](#) or all of the entity's key fields. The `id` control information MUST appear if any of the entity's key fields are omitted from the response or the entity-id is not identical to the canonical URL of the entity. For [ordered payloads](#), the control information `id`, if present, MUST immediately follow the control information [removed](#).

For full metadata the [context](#) control information MUST be included. It also MUST be included if the entity set of the deleted entity cannot be determined from the surrounding context.

The deleted-entity object MAY include additional properties of the entity, as well as [annotations](#), and MAY include related entities, related deleted entities, or a delta or full representation of a related collection of entities, to represent related entities that have been modified or deleted.

Example 37: deleted entity in OData 4.01 response with `id` control information (prefixed with an @)


```
{
  "@context": "#Customers/$deletedEntity",
  "@removed": {
    "reason": "deleted",
    "@myannoation.deletedBy": "Mario"
  },
  "@id": "Customers('ANTON') "
}
```

Example 38: entity removed OData 4.01 response without id control information and instead all key fields (ID is the single key field of Customer)

```
{
  "@removed": {},
  "ID": "ANTON"
}
```

15.4 Added Link

Links within a delta response are represented as link objects.

Delta responses MUST contain a link object for each added link that corresponds to a `$expand` path in the initial request.

The link object MUST include the following properties, regardless of the specified [metadata](#) value, and MAY include [annotations](#):

- [context](#) - the context URL fragment MUST be `#{entity-set}/$link`, where `{entity-set}` is the entity set containing the source entity
- `source` — The [id](#) of the entity from which the relationship is defined, which may be absolute or [relative](#)
- `relationship` — The path from the source object to the navigation property which MAY traverse one or more complex properties, type cast segments, or members of ordered collections
- `target` — The [id](#) of the related entity, which may be absolute or [relative](#)

15.5 Deleted Link

Deleted links within a delta response are represented as deleted-link objects.

Delta responses MUST contain a deleted-link object for each deleted link that corresponds to a `$expand` path in the initial request, unless either of the following is true:

- The `source` or `target` entity has been deleted
- The maximum cardinality of the related entity is one and there is a subsequent [link object](#) that specifies the same `source` and `relationship`.

The deleted-link object MUST include the following properties, regardless of the specified [metadata](#) value, and MAY include [annotations](#):

- [context](#) — the context URL fragment MUST be `#{entity-set}/$deletedLink`, where `{entity-set}` is the entity set containing the source entity
- `source` — The [id](#) of the entity from which the relationship is defined, which may be absolute or [relative](#)
- `relationship` — The path from the source object to the navigation property which MAY traverse one or more complex properties, type cast segments, or members of ordered collections
- `target` — The [id](#) of the related entity for multi-valued navigation properties, which may be absolute or [relative](#). For delta payloads that do not specify an `odata-version` header value of 4.0, the target MAY be omitted for single-valued navigation.

15.6 Update a Collection of Entities

The body of a PATCH request to a URL identifying a collection of entities is a JSON object. It MUST contain the [context](#) control information with a string value of `#$delta`, and it MUST contain an array-valued property named `value` containing all [added](#), [changed](#), or [deleted](#) entities, as well as [added](#) or [deleted](#) links between entities.

Example 39: 4.01 collection-update request for customers with expanded orders represented inline as a delta

1. Add customer EASTC
2. Change `ContactName` of customer AROUT
3. Delete customer ANTON
4. Change customer ALFKI:
 1. Create order 11011
 2. Add link to existing order 10692
 3. Change `RequiredDate` of related order 10835
 4. Delete link to order 10643
5. Add link between customer ANATR and order 10643
6. Delete link between customer DUMON and order 10311

```
PATCH /service/Customers HTTP/1.1
Host: host
Content-Type: application/json
Content-Length: ###
Prefer: return=minimal, continue-on-error

{
  "@context": "#$delta",
  "value": [
    {
      "@Org.OData.Core.V1.ContentID": "1",
      "CustomerID": "EASTC",
      "CompanyName": "Eastern Connection",
      "ContactName": "Ann Devon",
      "ContactTitle": "Sales Agent"
    },
    {
      "@Org.OData.Core.V1.ContentID": "2",
      "CustomerID": "AROUT",
      "ContactName": "Thomas Hardy",
    },
    {
      "@Org.OData.Core.V1.ContentID": "3",
      "@removed": {},
      "CustomerID": "ANTON"
    },
    {
      "@Org.OData.Core.V1.ContentID": "4",
      "CustomerID": "ALFKI",
      "Orders@delta": [
        {
          "@Org.OData.Core.V1.ContentID": "4.1",
          "OrderID": 11011,
          "CustomerID": "ALFKI",
          "EmployeeID": 3,
          "OrderDate": "1998-04-09T00:00:00Z",
          "RequiredDate": "1998-05-07T00:00:00Z",
          "ShippedDate": "1998-04-13T00:00:00Z"
        },
        {
          "@Org.OData.Core.V1.ContentID": "4.2",
```

```

    "@id": "Orders(10692)"
  },
  {
    "@Org.OData.Core.V1.ContentID": "4.3",
    "@id": "Orders(10835)",
    "RequiredDate": "1998-01-23T00:00:00Z",
  },
  {
    "@Org.OData.Core.V1.ContentID": "4.4",
    "@removed": {
      "reason": "changed"
    },
    "OrderID": 10643
  }
]
},
{
  "@context": "#Customers/$link",
  "@Org.OData.Core.V1.ContentID": "5",
  "source": "Customers('ANATR')",
  "relationship": "Orders",
  "target": "Orders(10643)"
},
{
  "@context": "#Customers/$deletedLink",
  "@Org.OData.Core.V1.ContentID": "6",
  "source": "Customers('DUMON')",
  "relationship": "Orders",
  "target": "Orders(10311)"
}
]
}

```

Assuming all changes can be applied without errors, the response would be

```

HTTP/1.1 204 No Content
Preference-Applied: return=minimal, continue-on-error

```

Assuming some or all changes cannot be applied, the overall request is still deemed successful due to the `continue-on-error` preference, and the response details what went wrong

1. Add customer 'EASTC' - failed
2. Change `ContactName` of customer 'AROUT' - failed
3. Delete customer 'ANTON' - failed
4. Change customer 'ALFKI':
 1. Create order 11011 - succeeded, not mentioned in response
 2. Add link to existing order 10692 - succeeded, not mentioned in response
 3. Change `RequiredDate` of related order 10835 - failed
 4. Delete link to order 10643 - succeeded, not mentioned in response
5. Add link between customer 'ANATR' and order 10643 - failed without further info
6. Delete link between customer 'DUMON' and order 10311 - failed without further info

```

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: ###
Preference-Applied: return=minimal, continue-on-error

{
  "@context": "#$delta",

```

```

"value": [
  {
    "@Org.OData.Core.V1.ContentID": "1",
    "CustomerID": "EASTC",
    "@removed": {},
    "@Org.OData.Core.V1.DataModificationException": {
      "failedOperation": "insert",
      "responseCode": 400,
      "info": {
        "code": "incmplt",
        "message": "Required field(s) not provided",
        "target": "Address",
        "@OtherVocab.additionalTargets": [ "Industry", "VATRegistration" ],
        "severity": "error"
      }
    }
  },
  {
    "@Org.OData.Core.V1.ContentID": "2",
    "CustomerID": "AROUT",
    "@Org.OData.Core.V1.DataModificationException": {
      "failedOperation": "update",
      "responseCode": 400,
      "info": {
        "code": "r-o",
        "message": "Customer is archived and cannot be changed",
        "severity": "error"
      }
    }
  },
  {
    "@Org.OData.Core.V1.ContentID": "3",
    "CustomerID": "ANTON",
    "@Org.OData.Core.V1.DataModificationException": {
      "failedOperation": "delete",
      "responseCode": 400,
      "info": {
        "code": "ufo",
        "message": "Customer has unfinished orders and cannot be deleted",
        "severity": "error"
      }
    }
  },
  {
    "@Org.OData.Core.V1.ContentID": "4",
    "CustomerID": "ALFKI",
    "Orders@delta": [
      {
        "@Org.OData.Core.V1.ContentID": "4.3",
        "@id": "Orders(10835)",
        "@Org.OData.Core.V1.DataModificationException": {
          "failedOperation": "update",
          "responseCode": 400,
          "info": {
            "code": "b/s",
            "message": "RequiredDate cannot be changed because Order is already being shipped",
            "severity": "error"
          }
        }
      }
    ]
  }
]

```

```

    ]
  },
  {
    "@context": "#Customers/$deletedLink",
    "@Org.OData.Core.V1.ContentID": "5",
    "source": "Customers('ANATR')",
    "relationship": "Orders",
    "target": "Orders(10643)",
    "@Org.OData.Core.V1.DataModificationException": {
      "failedOperation": "link",
      "responseCode": 404,
      "info": null
    }
  }
],
{
  "@context": "#Customers/$link",
  "@Org.OData.Core.V1.ContentID": "6",
  "source": "Customers('DUMON')",
  "relationship": "Orders",
  "target": "Orders(10311)",
  "@Org.OData.Core.V1.DataModificationException": {
    "failedOperation": "unlink",
    "responseCode": 400
  }
}
]
}

```

Without the continue-on-error preference processing would stop on the first error, and the response would be a standard OData error response

```

HTTP/1.1 400 Bad Request
Content-Type: application/json
Content-Length: ###

{
  "error": {
    "code": "incmplt",
    "message": "Required field(s) not provided",
    "target": "Customers('EASTC')/Address",
    "@OtherVocab.additionalTargets": [ "Customers('EASTC')/Industry",
      "Customers('EASTC')/VATRegistration" ]
  }
}

```

16 Bound Function

A bound function is advertised via a name/value pair where the name is a hash (#) character followed by the namespace- or alias-qualified name of the function. The namespace or alias MUST be defined or the namespace referenced in the metadata document of the service, see [\[OData-CSDLJSON\]](#) or [\[OData-CSDLXML\]](#). A specific function overload can be advertised by appending the parentheses-enclosed, comma-separated list of non-binding parameter names to the qualified function name, see rule `qualifiedFunctionName` in [\[OData-ABNF\]](#).

A function that is bound to a single structured type MAY be advertised within the JSON object representing that structured type.

Functions that are bound to a collection MAY be advertised within the JSON object containing the collection. If the collection is the top-level response, the function advertisement name/value pair is placed next to the `value` name/value pair representing the collection. If the collection is nested within an instance of a structured type, then in 4.01 payloads the name of the function advertisement is prepended with the name of the collection-valued property and is placed next to the collection-valued property, [expanded navigation property](#), or [navigationLink](#) control information, if present. 4.0 payloads MUST NOT advertise functions prefixed with property names.

If the function is available, the value of the advertisement is an object. OData 4.01 services MAY advertise the non-availability of the function with the value `null`.

If [metadata=full](#) is requested, each value object MUST have at least the two name/value pairs `title` and `target`. It MAY contain [annotations](#). The order of the name/value pairs MUST be considered insignificant.

The `target` name/value pair contains a URL. Clients MUST be able to invoke the function or the specific function overload by passing the parameter values via query options for [parameter aliases](#) that are identical to the parameter name preceded by an at (@) sign. Clients MUST check if the obtained URL already contains a query part and appropriately precede the parameters either with an ampersand (&) or a question mark (?).

The `title` name/value pair contains the function or action title as a string.

If [metadata=minimal](#) is requested, the `target` name/value pair MUST be included if its value differs from the canonical function or action URL.

Example 40: minimal representation of a function where all overloads are applicable

```
{
  "@context": "http://host/service/$metadata#Employees/$entity",
  "#Model.RemainingVacation": {},
  ...
}
```

Example 41: full representation of a specific overload with parameter alias for the Year parameter

```
{
  "@context": "http://host/service/$metadata#Employees/$entity",
  "#Model.RemainingVacation(Year)": {
    "title": "Remaining vacation from year.",
    "target": "Employees(2)/RemainingVacation(Year=@Year)"
  },
  ...
}
```

Example 42: full representation in a collection

```
{
  "@context": "http://host/service/$metadata#Employees",
  ...
}
```

```
"#Model.RemainingVacation": {  
  "title": "Remaining Vacation",  
  "target": "Managers(22)/Employees/RemainingVacation"  
},  
"value": [ ... ]  
}
```

Example 43: full representation in a nested collection

```
{  
  "@context": "http://host/service/$metadata#Employees/$entity",  
  "@type": "Model.Manager",  
  "ID": 22,  
  ...  
  "Employees#RemainingVacation": {  
    "title": "RemainingVacation",  
    "target": "Managers(22)/Employees/RemainingVacation"  
  }  
}
```

17 Bound Action

A bound action is advertised via a name/value pair where the name is a hash (#) character followed by the namespace- or alias-qualified name of the action. The namespace or alias MUST be defined or the namespace referenced in the metadata document of the service, see [\[OData-CSDLJSON\]](#) or [\[OData-CSDLXML\]](#)

An action that is bound to a single structured type is advertised within the JSON object representing that structured type.

Actions that are bound to a collection MAY be advertised within the JSON object containing the collection. If the collection is the top-level response, the action advertisement name/value pair is placed next to the `value` name/value pair representing the collection. If the collection is nested within an instance of a structured type, then in 4.01 payloads the name of the action advertisement is prepended with the name of the collection-valued property and is placed next to the name/value pair representing the collection-valued property, [expanded navigation property](#), or [navigationLink](#) control information, if present. 4.0 payloads MUST NOT advertise actions prefixed with property names.

If the action is available, the value of the advertisement is an object. OData 4.01 services MAY advertise the non-availability of the action with the value `null`.

If [metadata=full](#) is requested, each value object MUST have at least the two name/value pairs `title` and `target`. It MAY contain [annotations](#). The order of these name/value pairs MUST be considered insignificant.

The `target` name/value pair contains a bound function or action URL.

The `title` name/value pair contains the function or action title as a string.

If [metadata=minimal](#) is requested, the `target` name/value pair MUST be included if its value differs from the canonical function or action URL.

Example 44: minimal representation in an entity

```
{
  "@context": "http://host/service/$metadata#LeaveRequests/$entity",
  "#Model.Approve": {},
  ...
}
```

Example 45: full representation in an entity:

```
{
  "@context": "http://host/service/$metadata#LeaveRequests/$entity",
  "#Model.Approve": {
    "title": "Approve Leave Request",
    "target": "LeaveRequests(2)/Approve"
  },
  ...
}
```

Example 46: full representation in a collection

```
{
  "@context": "http://host/service/$metadata#LeaveRequests",
  "#Model.Approve": {
    "title": "Approve All Leave Requests",
    "target": "Employees(22)/Model.Manager/LeaveRequests/Approve"
  },
  ...
}
```



```
"value": [ ... ]  
}
```

Example 47: full representation in a nested collection

```
{  
  "@context": "http://host/service/$metadata#Employees/$entity",  
  "@type": "Model.Manager",  
  "ID": 22,  
  ...  
  "LeaveRequests#Model.Approve": {  
    "title": "Approve All Leave Requests",  
    "target": "Employees(22)/Model.Manager/LeaveRequests/Approve"  
  }  
}
```

18 Action Invocation

Action parameter values are encoded in a single JSON object in the request body.

Each non-binding parameter value is encoded as a separate name/value pair in this JSON object. The name is the name of the parameter. The value is the parameter value in the JSON representation appropriate for its type. Entity typed parameter values MAY include a subset of the properties, or just the [entity reference](#), as appropriate to the action. Stream typed parameter values are represented following the same rules as inlined [stream properties](#).

Non-binding parameters that are nullable or annotated with the term [Core.OptionalParameter](#) defined in [\[OData-VocCore\]](#) MAY be omitted from the request body. If an omitted parameter is not annotated (and thus nullable), it MUST be interpreted as having the `null` value. If it is annotated and the annotation specifies a `DefaultValue`, the omitted parameter is interpreted as having that default value. If omitted and the annotation does not specify a default value, the service is free on how to interpret the omitted parameter. Note: a nullable non-binding parameter is equivalent to being annotated as optional with a default value of `null`.

Example 46:

```
{
  "param1": 42,
  "param2": {
    "Street": "One Microsoft Way",
    "Zip": 98052
  },
  "param3": [ 1, 42, 99 ],
  "param4": null
}
```

In order to invoke an action with no non-binding parameters, the client passes an empty JSON object in the body of the request. 4.01 Services MUST also support clients passing an empty request body for this case.

19 Batch Requests and Responses

19.1 Batch Request

A JSON batch request body consists of a single JSON object that **MUST** contain the name/value pair `requests` and **MAY** contain [annotations](#). It does not contain the `context` control information.

The value of `requests` is an array of request objects, each representing an individual request. Note: an individual request **MUST NOT** itself be a batch request.

A *request object* **MUST** contain the name/value pairs `id`, `method` and `url`, and it **MAY** contain the name/value pairs `atomicityGroup`, `dependsOn`, `if`, `headers`, and `body`.

The value of `id` is a string containing the request identifier of the individual request, see [\[OData-Protocol\]](#). It **MUST NOT** be identical to the value of any other request identifier nor any `atomicityGroup` within the batch request.

Note: the `id` name/value pair corresponds to the `Content-ID` header in the multipart batch format specified in [\[OData-Protocol\]](#).

The value of `method` is a string that **MUST** contain one of the literals `delete`, `get`, `patch`, `post`, or `put`. These literals are case-insensitive.

The value of `url` is a string containing the individual request URL. The URL **MAY** be an absolute path (starting with a forward slash /) which is appended to scheme, host, and port of the batch request URL, or a relative path (not starting with a forward slash /).

If the first segment of a relative path starts with a `$` character and is not identical to the name of a top-level system resource (`$batch`, `$crossjoin`, `$all`, `$entity`, `$root`, `$id`, `$metadata`, or other system resources defined according to the *OData-Version* of the protocol specified in the request), then this first segment is replaced with the URL of the entity created by or returned from a preceding request whose `id` value is identical to the value of the first segment with the leading `$` character removed. The `id` of this request **MUST** be specified in the `dependsOn` name/value pair.

Otherwise, the relative path is resolved relative to the batch request URL (i.e. relative to the service root).

The value of `atomicityGroup` is a string whose content **MUST NOT** be identical to any value of `id` within the batch request, and which **MUST** satisfy the rule `request-id` in [\[OData-ABNF\]](#). All request objects with the same value for `atomicityGroup` **MUST** be adjacent in the `requests` array. These requests are processed as an atomic operation and **MUST** either all succeed, or all fail.

Note: the atomicity group is a generalization of the change set in the multipart batch format specified in [\[OData-Protocol\]](#).

The value of `dependsOn` is an array of strings whose values **MUST** be values of either `id` or `atomicityGroup` of preceding request objects; forward references are not allowed. If a request depends on another request that is part of a different atomicity group, the atomicity group **MUST** be listed in `dependsOn`. In the absence of the optional `if` member a request that depends on other requests or atomicity groups is only executed if those requests were executed successfully, i.e. with a `2xx` response code. If one of the requests it depends on has failed, the dependent request is not executed and a response with status code of `424 Failed Dependency` is returned for it as part of the batch response.

The `if` member can specify an alternative condition for executing the dependent request. Its value **MUST** be URL expression (see [\[OData-URL\]](#)) that evaluates to a Boolean value. The URL expression syntax is extended and additionally allows

- `$<content-id>/$succeeded` to check if the referenced request has succeeded
- `$<content-id>` to reference the response body of the referenced request
- `$<content-id>/<path>` to reference a part of the response body

Services SHOULD advertise support of the `if` member by specifying the property `RequestDependencyConditionsSupported` in the [Capabilities.BatchSupport](#) term applied to the entity container, see [\[OData-VocCap\]](#). If a service does not support request dependencies, the dependent request MUST fail with 424 Failed Dependency, and if the dependent request is part of an atomicity group, all requests in that group fail with 424 Failed Dependency with no changes applied.

The value of `headers` is an object whose name/value pairs represent request headers. The name of each pair MUST be the lower-case header name; the value is a string containing the header-encoded value of the header. The `headers` object MUST contain a name/value pair with the name `content-type` whose value is the media type.

The value of `body` can be `null`, which is equivalent to not specifying the `body` name/value pair.

For media type `application/json` or one of its subtypes, optionally with format parameters, the value of `body` is JSON.

For media types of top-level type `text`, for example `text/plain`, the value of `body` is a string containing the value of the request body.

For all other media types the value of `body` is a string containing the base64url-encoded value of the request body. In this case the body content can be compressed or chunked if this is correctly reflected in the `Transfer-Encoding` header.

A body MUST NOT be specified if the `method` is `get` or `delete`.

The request object and the `headers` object MUST NOT contain name/value pairs with duplicate names. This is in conformance with [\[RFC7493\]](#).

Example 48: a batch request that contains the following individual requests in the order listed

1. A query request
2. An atomicity group that contains the following requests:
 - Insert entity
 - Update entity
3. A second query request

Note: For brevity, in the example, request bodies are excluded in favor of English descriptions inside <> brackets and `OData-Version` headers are omitted.

```
POST /service/$batch HTTP/1.1
Host: host
OData-Version: 4.01
Content-Type: application/json
Content-Length: ###

{
  "requests": [
    {
      "id": "0",
      "method": "get",
      "url": "/service/Customers('ALFKI')"
    },
    {
      "id": "1",
      "atomicityGroup": "group1",
      "dependsOn": [ "0" ],
      "method": "patch",
      "url": "/service/Customers('ALFKI')",
      "headers": {
        "Prefer": "return=minimal"
      }
    },
  ],
}
```

```

    "body": <JSON representation of changes to Customer ALFKI>
  },
  {
    "id": "2",
    "atomicityGroup": "group1",
    "method": "post",
    "url": "/service/Customers",
    "body": <JSON representation of a new Customer entity>
  },
  {
    "id": "3",
    "dependsOn": [ "group1" ],
    "method": "get",
    "url": "/service/Products"
  }
]
}

```

19.2 Referencing New Entities

The entity returned by a preceding request can be referenced in the request URL of subsequent requests. If the **Location** header in the response contains a relative URL, clients **MUST** be able to resolve it relative to the request's URL even if that contains such a reference.

Example 49: a batch request that contains the following operations in the order listed:

- Insert a new entity (with id = 1)
- Insert a second new entity (references request with id = 1)

```

POST /service/$batch HTTP/1.1
Host: host
OData-Version: 4.01
Content-Type: application/json
Content-Length: ###

{
  "requests": [
    {
      "id": "1",
      "method": "post",
      "url": "/service/Customers",
      "body": <JSON representation of a new Customer entity>
    },
    {
      "id": "2",
      "dependsOn": [ "1" ]
      "method": "post",
      "url": "$1/Orders",
      "body": <JSON representation of a new Order>
    }
  ]
}

```

19.3 Referencing an ETag

Example 50: a batch request that contains the following operations in the order listed:

- Get an Employee (with id = 1)
- Update the salary only if the employee has not changed

```

POST /service/$batch HTTP/1.1
Host: host
OData-Version: 4.01
Content-Type: application/json
Content-Length: ###

{
  "requests": [
    {
      "id": "1",
      "method": "get",
      "url": "/service/Employees(0)",
      "headers": {
        "accept": "application/json"
      }
    },
    {
      "id": "2",
      "dependsOn": [ "1" ],
      "method": "patch",
      "url": "/service/Employees(0)",
      "headers": {
        "if-match": "$1"
      },
      "body": {
        "Salary": 75000
      }
    }
  ]
}

```

19.4 Referencing Response Body Values

Example 51: a batch request that contains the following operations in the order listed:

- Get an employee (with Content-ID = 1)
- Get all employees residing in the same building

```

POST /service/$batch HTTP/1.1
Host: host
OData-Version: 4.01
Content-Type: application/json
Content-Length: ###

{
  "requests": [
    {
      "id": "1",
      "method": "get",
      "url": "/service/Employees/0?$select=Building",
      "headers": {
        "accept": "application/json"
      }
    },
    {
      "id": "2",
      "dependsOn": [ "1" ],
      "method": "get",
      "url": "/service/Employees?$filter=Building eq $1/Building",
      "headers": {

```

```

    "accept": "application/json"
  }
}
]
}

```

19.5 Processing a Batch Request

All requests in an atomicity group represent a single change unit. A service **MUST** successfully process and apply all the requests in the atomicity group or else apply none of them. It is up to the service implementation to define rollback semantics to undo any requests within an atomicity group that may have been applied before another request in that same atomicity group failed.

The service **MAY** process the individual requests and atomicity groups within a batch request, or individual requests within an atomicity group, in any order that is compatible with the dependencies expressed with the `dependsOn` name/value pair. Individual requests and atomicity groups that do not specify the `dependsOn` name/value pair may be processed in parallel. Clients that are only interested in completely successful batch responses **MAY** specify the preference `continue-on-error=false` to indicate that the service need not spend cycles on further processing once an error occurs in one of the dependency chains. In this case the response **MAY** omit response objects for requests that have not been processed. If the preference `continue-on-error` is not specified, or specified with a value of `true`, all requests are processed according to their dependencies.

The service **MUST** include the `id` name/value pair in each response object with the value of the request identifier that the client specified in the corresponding request, so clients can correlate requests and responses.

19.6 Batch Response

A JSON batch response body consists of a single JSON object that **MUST** contain the name/value pair `responses` and **MAY** contain [annotations](#). It does not contain the `context` control information.

The value of `responses` is an array of response objects, each representing an individual response.

A JSON batch response **MAY** be a partial result containing the [nextLink](#) control information. This allows services to chunk results into manageable pieces, or to return results for already processed requests and continue processing the remaining individual requests while waiting for the client to fire a `GET` request to the next link.

In a response to a batch request using the multipart format defined in [\[OData-Protocol\]](#) the response objects **MUST** appear in the same order as required for multipart batch responses because the `Content-ID` header is not required outside of change sets. Response objects corresponding to requests that specify a `Content-ID` header **MUST** contain the `id` name/value pair, and the value of `id` **MUST** be the value of the `Content-ID` header of the corresponding request. This is necessarily the case for requests contained within a change set. Responses to requests within a change set **MUST** contain the `atomicityGroup` name/value pair with a value common within a change set and unique across change sets.

In a response to a batch request using the JSON batch request format specified in the preceding section the response objects **MAY** appear in any order, and each response object **MUST** contain the `id` name/value pair with the same value as in the corresponding request object. If the corresponding request object contains the `atomicityGroup` name/value pair, it **MUST** also be present in the response object with the same value.

If any response within an atomicity group returns a failure code, all requests within that atomicity group are considered failed, regardless of their individual returned status code. The service **MAY** return `424 Failed Dependency` for statements within an atomicity group that fail or are not attempted due to other failures within the same atomicity group.

A response object **MUST** contain the name/value pair `status` whose value is a number representing the HTTP status code of the response to the individual request.

The response object MAY contain the name/value pair `headers` whose value is an object with name/value pairs representing response headers. The name of each pair MUST be the lower-case header name; the value is a string containing the header-encoded value of the header.

The response object MAY contain the name/value pair `body` which follows the same rules as within [request objects](#).

If the media type is not exactly equal to `application/json` (i.e. it is a subtype or has format parameters), the `headers` object MUST contain a name/value pair with the name `content-type` whose value is the media type.

Relative URLs in a response object follow the rules for [relative URLs](#) based on the request URL of the corresponding request. Especially: URLs in responses MUST NOT contain `$`-prefixed request identifiers.

Example 52: referencing the batch request [example 48](#) above, assume all the requests except the final query request succeed. In this case the response would be

```
HTTP/1.1 200 OK
OData-Version: 4.01
Content-Length: #####
Content-Type: application/json

{
  "responses": [
    {
      "id": "0",
      "status": 200,
      "body": <JSON representation of the Customer entity with key ALFKI>
    },
    {
      "id": "1",
      "status": 204
    },
    {
      "id": "2",
      "status": 201,
      "headers": {
        "location": "http://host/service.svc/Customer('POIUY') "
      },
      "body": <JSON representation of the new Customer entity>
    },
    {
      "id": "3",
      "status": 404,
      "body": <Error message>
    }
  ]
}
```

19.7 Asynchronous Batch Requests

A batch request that specifies the `respond-async` preference MAY be executed asynchronously. This means that the “outer” batch request is executed asynchronously; this preference does not automatically cascade down to the individual requests within the batch. After successful execution of the batch request the response to the batch request is returned in the body of a response to an interrogation request against the status monitor resource URL, see section “Asynchronous Requests” in [\[OData-Protocol\]](#).

A service MAY return interim results to an asynchronously executing batch. It does this by responding with 200 OK to a GET request to the monitor resource and including a [nextLink](#) control information in the JSON batch response, thus signaling that the response is only a partial result. A subsequent GET request to the next link MAY result in a 202 Accepted response with a `location` header pointing to a new status monitor resource.

Example 53: referencing the example 47 above again, assume that the request is sent with the `respond-async` preference. This results in a 202 response pointing to a status monitor resource:

```
HTTP/1.1 202 Accepted
Location: http://service-root/async-monitor-0
Retry-After: ###
```

When interrogating the monitor URL only the first request in the batch has finished processing and all the remaining requests are still being processed. The service signals that asynchronous processing is “finished” and returns a partial result with the first response and a next link. The client did not explicitly accept `application/http`, so the response is “unwrapped” and only indicates with the `AsyncResult` header that it is a response to a status monitor resource:

```
HTTP/1.1 200 OK
AsyncResult: 200
OData-Version: 4.01
Content-Length: ###
Content-Type: application/json

{
  "responses": [
    {
      "id": "0",
      "status": 200,
      "body": <JSON representation of the Customer entity with key ALFKI>
    }
  ],
  "@nextLink": "...?$skiptoken=YmF0Y2gx"
}
```

Client makes a GET request to the next link and receives a 202 response with the location of a new monitor resource.

```
HTTP/1.1 202 Accepted
Location: http://service-root/async-monitor-1
Retry-After: ###
```

After some time a GET request to the monitor resource returns the remainder of the result.

```
HTTP/1.1 200 OK
AsyncResult: 200
OData-Version: 4.01
Content-Length: ###
Content-Type: application/json

{
  "responses": [
    {
      "id": "1",
      "status": 204
    },
    {
      "id": "2",
      "status": 201,
      "headers": {
        "location": "http://host/service.svc/Customer('POIUY')"
      },
      "body": <JSON representation of the new Customer entity>
    },
    {
      "id": "3",
      "status": 404,
    }
  ]
}
```

```

    "body": <Error message>
  }
]
}

```

In addition to the above interaction pattern individual requests within a batch with no other requests depending on it and not part of an atomicity group MAY be executed asynchronously if they specify the `respond-async` preference and if the service responds with a JSON batch response. In this case the `response` array contains a response object for each asynchronously executed individual request with a `status` of 202, a `location` header pointing to an individual status monitor resource, and optionally a `retry-after` header.

Example 54: the first individual request is processed asynchronously, the second synchronously, the batch itself is processed synchronously

```

HTTP/1.1 200 OK
OData-Version: 4.01
Content-Length: ###
Content-Type: application/json

{
  "responses": [
    {
      "id": "0",
      "status": 202,
      "headers": {
        "location": "http://service-root/async-monitor-0"
      }
    },
    {
      "id": "1",
      "status": 204
    }
  ]
}

```

20 Instance Annotations

Annotations are an extensibility mechanism that allows services and clients to include information other than the raw data in the request or response.

Annotations are name/value pairs that have an at (@) and a dot (.) as part of the name. The part after the “at” sign (@) is the *annotation identifier*. It consists of the namespace or alias of the schema that defines the term, followed by a dot (.), followed by the name of the term, optionally followed by a hash (#) and a qualifier. The namespace or alias MUST be defined in the metadata document, see [\[OData-CSDLJSON\]](#) or [\[OData-CSDLXML\]](#)

The annotation identifier `odata` is reserved for future extensions of the protocol and format. Instance annotations MUST have a namespace or alias that is different from `odata`.

Annotations can be applied to any name/value pair in a JSON payload that represents a value of any type from the entity data model. Clients should never error due to an unexpected annotation in a JSON payload.

Annotations are always expressed as name/value pairs. For entity data model constructs represented as JSON objects the annotation name/value pairs are placed within the object; for constructs represented as JSON arrays or primitives, including null, they are placed next to the annotated model construct and have the name of the annotated property before the @. An annotation in the latter format can also take the place of an absent property. When annotating a payload that represents a [single primitive or collection value](#), the annotations for the value appear next to the `value` property and are not prefixed with a property name.

Example 55:

```
{
  "@context": "http://host/service/$metadata#Customers",
  "@com.example.customer.setkind": "VIPs",
  "value": [
    {
      "@com.example.display.highlight": true,
      "ID": "ALFKI",
      "CompanyName@com.example.display.style": { "title": true, "order": 1 },
      "CompanyName": "Alfreds Futterkiste",
      "Orders@com.example.display.style#simple": { "order": 2 }
    }
  ]
}
```

20.1 Annotate a JSON Object

When annotating a name/value pair for which the value is represented as a JSON object, each annotation is placed within the object and represented as a single name/value pair.

The name always starts with the “at” sign (@), followed by the annotation identifier.

The value MUST be an appropriate value for the annotation.

20.2 Annotate a JSON Array or Primitive

When annotating a name/value pair for which the value is represented as a JSON array or primitive value, each annotation that applies to this name/value pair MUST be represented as a single name/value pair and placed immediately prior to the annotated name/value pair, with the exception of the [nextLink](#) or [collectionAnnotations](#) control information, which can appear immediately before or after the annotated collection.

The name is the same as the name of the property or name/value pair being annotated, followed by the “at” sign (@), followed by the annotation identifier.

The value MUST be an appropriate value for the annotation.

20.3 Annotate a Primitive Value within a JSON Array

Individual primitive elements within a JSON array can be annotated by applying the [collectionAnnotations](#) control information to the array containing the primitive member.

The control information must come with other annotations or control information immediately before or after the collection valued property. The name of the property representing the control information is the same as the name of the collection-valued property, followed by the “at” sign (@), followed by the [collectionAnnotations](#) identifier.

21 Error Handling

OData requests may return a well formed [error response](#), an [in-stream error](#), or error information [within a success payload](#).

21.1 Error Response

The error response MUST be a single JSON object. This object MUST have a single name/value pair named `error`. The value must be an OData error object.

The OData error object MUST contain name/value pairs with the names `code` and `message`, and it MAY contain name/value pairs with the names `target`, `details`, and `innererror`.

The value for the `code` name/value pair is a non-empty language-independent string. Its value is a service-defined error code. This code serves as a sub-status for the HTTP error code specified in the response. It cannot be `null`.

The value for the `message` name/value pair is a non-empty, language-dependent, human-readable string describing the error. The `Content-Language` header MUST contain the language code from [\[RFC5646\]](#) corresponding to the language in which the value for message is written. It cannot be `null`.

The value for the `target` name/value pair is a potentially empty string indicating the target of the error (for example, the name of the property in error). It can be `null`.

The value for the `details` name/value pair MUST be an array of JSON objects that MUST contain name/value pairs for `code` and `message`, and MAY contain a name/value pair for `target`, as described above.

The value for the `innererror` name/value pair MUST be an object. The contents of this object are service-defined. Usually this object contains information that will help debug the service.

Service implementations SHOULD carefully consider which information to include in production environments to guard against potential security concerns around information disclosure.

Error responses MAY contain [annotations](#) in any of its JSON objects.

Example 56:

```
{
  "error": {
    "code": "err123",
    "message": "Unsupported functionality",
    "target": "query",
    "details": [
      {
        "code": "forty-two",
        "target": "$search",
        "message": "$search query option not supported"
      }
    ],
    "innererror": {
      "trace": [...],
      "context": {...}
    }
  }
}
```

21.2 In-Stream Error

In the case that a service encounters an error after sending a success status to the client, the service **MUST** leave the response malformed. This can be achieved by immediately stopping response serialization and thus omitting (among others) the end-object character of the top-level JSON object in the response.

Services **MAY** include the header `OData-Error` as a trailing header if supported by the transport protocol (e.g. with HTTP/1.1 and chunked transfer encoding, or with HTTP/2), see [\[OData-Protocol\]](#).

The value of the `OData-Error` trailing header is an OData error object as defined in the preceding chapter, represented in a header-appropriate way:

- All optional whitespace (indentation and line breaks) is removed, especially (in hex notation) 09, 0A and 0D
- Control characters (00 to 1F and 7F) and Unicode characters beyond 00FF within JSON strings are encoded as `\uXXXX` or `\uXXXX\uXXXX` (see [\[RFC8259\]](#), section 7)

Example 57: note that this is one HTTP header line without any line breaks or optional whitespace

```
OData-error: {"code":"err123","message":"Unsupported
functionality","target":"query","details":[{"code":"forty-
two","target":"$search","message":"$search
query option not supported"}]}
```

21.3 Error Information in a Success Payload

Services may return error information within a success payload; for example, if the client has specified the `continue-on-error` preference.

21.3.1 Primitive Value Errors

Primitive values that are in error are annotated with the `Core.ValueException` term, see [\[OData-VocCore\]](#). In this case, the payload **MAY** include an approximation of the value and **MAY** specify a string representation of the exact value in the `value` property of the annotation.

21.3.2 Structured Type Errors

Structured types that are in error can be represented within a success payload only if the client has specified the `continue-on-error` preference. Such items are annotated with the `Core.ResourceException` term, see [\[OData-VocCore\]](#). The annotation **MAY** include a `retryLink` property that can be used by the client to attempt to re-fetch the resource.

21.3.3 Collection Errors

Collections within a success payload can contain primitive values that are in error, or structured values that are in error, if the client has specified the `continue-on-error` preference. Such elements are annotated as described above. Primitive elements within a collection are annotated using the [collectionAnnotations](#) control information.

Services can return partial collections within a success payload, for example, if they encounter an error while retrieving the collection and the client has specified the `continue-on-error` preference. In this case, the service **MUST** include a [nextLink](#). The `nextLink` can be used to attempt retrieving the remaining members of the collection and could return an error indicating that the remaining members are not available.

22 Extensibility

Implementations can add [instance annotations](#) of the form `@namespace.termname` Or `property@namespace.termname` to any JSON object, where `property` MAY or MAY NOT match the name of a name/value pair within the JSON object. However, the namespace MUST NOT start with `odata` and SHOULD NOT be required to be understood by the receiving party in order to correctly interpret the rest of the payload as the receiving party MUST ignore unknown annotations not defined in this version of the OData JSON Specification.

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", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005. <https://www.rfc-editor.org/info/rfc3986>.

[RFC3987]

Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, DOI 10.17487/RFC3987, January 2005. <https://www.rfc-editor.org/info/rfc3987>.

[RFC4648]

Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006. <https://www.rfc-editor.org/info/rfc4648>.

[RFC5646]

Phillips, A., Ed., and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009. <https://www.rfc-editor.org/info/rfc5646>.

[RFC7493]

Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015. <https://www.rfc-editor.org/info/rfc7493>.

[RFC7946]

Butler, H., Daly, M., Doyle, A., Gillies, S., Hagen, S., and T. Schaub, "The GeoJSON Format", RFC 7946, DOI 10.17487/RFC7946, August 2016. <https://www.rfc-editor.org/info/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", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017. <https://www.rfc-editor.org/info/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	Zurmuehl	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

Copyright © OASIS Open 2024. All Rights Reserved.

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.]

The name “OASIS” is a trademark of [OASIS](#), the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <https://www.oasis-open.org/policies-guidelines/trademark/> for above guidance.