OData JSON Format Version 4.0 Plus Errata 03 (redlined)

OASIS Standard incorporating Approved Errata 03

02 June 2016

Specification URIs
This version:
http://docs.oasis-open.org/odata/odata-json-format/v4.0/errata03/os/odata-json-format-v4.0-errata03-os-redlined.doc (Authoritative)
http://docs.oasis-open.org/odata/odata-json-format/v4.0/errata03/os/odata-json-format-v4.0-errata03-os-redlined.html
http://docs.oasis-open.org/odata/odata-json-format/v4.0/errata03/os/odata-json-format-v4.0-errata03-os-redlined.pdf

Previous version:
http://docs.oasis-open.org/odata/odata-json-format/v4.0/errata02/os/odata-json-format-v4.0-errata02-os-complete.doc (Authoritative)
http://docs.oasis-open.org/odata/odata-json-format/v4.0/errata02/os/odata-json-format-v4.0-errata02-os-complete.html
http://docs.oasis-open.org/odata/odata-json-format/v4.0/errata02/os/odata-json-format-v4.0-errata02-os-complete.pdf

Latest version:
http://docs.oasis-open.org/odata/odata-json-format/v4.0/errata03/os/odata-json-format-v4.0-errata03-complete.html (Authoritative)
http://docs.oasis-open.org/odata/odata-json-format/v4.0/errata03/os/odata-json-format-v4.0-errata03-complete.html
http://docs.oasis-open.org/odata/odata-json-format/v4.0/errata03/os/odata-json-format-v4.0-errata03-complete.pdf

Technical Committee:
OASIS Open Data Protocol (OData) TC

Chairs:
Ralf Handl (ralf.handl@sap.com), SAP SE
Ram Jeyaraman (Ram.Jeyaraman@microsoft.com), Microsoft

Editors:
Ralf Handl (ralf.handl@sap.com), SAP SE
Michael Pizzo (mikep@microsoft.com), Microsoft
Mark Biamonte (mark.biamonte@progress.com), Progress Software

Additional artifacts:
This prose specification is one component of a Work Product that also includes:
- OData JSON Format Version 4.0 Plus Errata 03. Edited by Ralf Handl, Michael Pizzo, and Mark Biamonte. 02 June 2016. OASIS Standard incorporating Approved Errata 03.
Related work:
This specification is related to:

- **OData Version 4.0**, a multi-part Work Product which includes:
  - Vocabulary components: OData Core Vocabulary, OData Measures Vocabulary and OData Capabilities Vocabulary. http://docs.oasis-open.org/odata/odata/v4.0/errata03/os/complete/vocabularies/.
  - OData Metadata Service Entity Model: http://docs.oasis-open.org/odata/odata/v4.0/errata03/os/complete/models/.

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 4.0 Part 1: Protocol; this document extends the former 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 version" 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/.

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).

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

[OData-JSON-Format-v4.0]
Notices

Copyright © OASIS Open 2016. 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.

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 Committee Specification or OASIS Standard, 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 specification.

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 specification 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 specification. 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 document 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 Committee Specification or OASIS Standard, 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.
# Table of Contents

1. Introduction .................................................................................................................. 7  
1.1 Terminology .................................................................................................................. 7  
1.2 Normative References ................................................................................................. 7  
1.3 Typographical Conventions .......................................................................................... 8  
2. JSON Format Design ...................................................................................................... 9  
3. Requesting the JSON Format ......................................................................................... 10  
3.1 Controlling the Amount of Control Information in Responses .................................. 10  
3.1.1 odata.metadata=minimal ......................................................................................... 10  
3.1.2 odata.metadata=full ............................................................................................... 11  
3.1.3 odata.metadata=none ............................................................................................. 11  
3.2 Controlling the Representation of Numbers .................................................................. 12  
4. Common Characteristics ................................................................................................. 13  
4.1 Header Content-Type .................................................................................................... 13  
4.2 Message Body ............................................................................................................... 13  
4.3 Relative URLs .............................................................................................................. 13  
4.4 Payload Ordering Constraints ...................................................................................... 14  
4.5 Control Information ...................................................................................................... 15  
4.5.1 Annotation odata.context ....................................................................................... 15  
4.5.2 Annotation odata.metadataEtag .............................................................................. 15  
4.5.3 Annotation odata.type .......................................................................................... 15  
4.5.4 Annotation odata.count ......................................................................................... 16  
4.5.5 Annotation odata.nextLink .................................................................................... 16  
4.5.6 Annotation odata.deltaLink ................................................................................... 16  
4.5.7 Annotation odata.id ............................................................................................. 17  
4.5.8 Annotation odata.editLink and odata.readLink .................................................... 17  
4.5.9 Annotation odata.etag ......................................................................................... 18  
4.5.10 Annotation odata.navigationLink and odata.associationLink ............................ 18  
4.5.11 Annotation odata.media* ..................................................................................... 18  
5. Service Document .......................................................................................................... 20  
6. Entity ............................................................................................................................... 22  
7. Structural Property ......................................................................................................... 23  
7.1 Primitive Value ............................................................................................................ 23  
7.2 Complex Value ............................................................................................................ 24  
7.3 Collection of Primitive Values ..................................................................................... 24  
7.4 Collection of Complex Values ..................................................................................... 24  
8. Navigation Property ....................................................................................................... 26  
8.1 Navigation Link ............................................................................................................ 26  
8.2 Association Link ........................................................................................................... 26  
8.3 Expanded Navigation Property ................................................................................... 26  
8.4 Deep Insert .................................................................................................................. 27  
8.5 Bind Operation ............................................................................................................. 27  
9. Stream Property .............................................................................................................. 29
10 Media Entity .................................................................................................................. 30
11 Individual Property or Operation Response .................................................................... 31
12 Collection of Entities ........................................................................................................ 32
13 Entity Reference ............................................................................................................... 33
14 Delta Response .................................................................................................................. 34
14.1 Added/Changed Entity ................................................................................................. 35
14.2 Deleted Entity ............................................................................................................... 35
14.3 Added Link .................................................................................................................... 35
14.4 Deleted Link .................................................................................................................. 36
15 Bound Function ............................................................................................................... 37
16 Bound Action .................................................................................................................... 38
17 Action Invocation .............................................................................................................. 39
18 Instance Annotations ....................................................................................................... 40
18.1 Annotate a JSON Object ............................................................................................. 40
18.2 Annotate a JSON Array or Primitive ......................................................................... 41
19 Error Response .................................................................................................................. 42
20 Extensibility ....................................................................................................................... 43
21 Security Considerations .................................................................................................. 44
22 Conformance .................................................................................................................... 45
Appendix A. Acknowledgments ............................................................................................ 46
Appendix B. Revision History .............................................................................................. 47
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 [RFC7159].

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 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.2 Normative References


See link in "Related work" section on cover page.


See link in "Related work" section on cover page.


See link in "Related work" section on cover page.


See link in "Related work" section on cover page.

[OData-VocCap] OData Capabilities Vocabulary.

See link in "Related work" section on cover page.


1.3 Typographical Conventions

Keywords defined by this specification use this monospaced font.

Normative source code uses this paragraph style.

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.
2 JSON Format Design

JSON, as described in [RFC7159], 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 annotations for control information such as ids, types, and links, and custom 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.

Annotations are used in JSON to capture control information 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 MIME 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 MIME type application/json, optionally followed by format parameters.

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

Possible format parameters are:

- odata.metadata
- IEEE754Compatible
- ExponentialDecimals
- odata.streaming

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

Services SHOULD advertise the supported MIME 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 odata.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., odata.streaming) are orthogonal to the odata.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 odata.metadata=minimal. If computation is more critical than wire size or the client is incapable of computing control information, odata.metadata=full directs the service to inline the control information that normally would be computed from metadata expressions in the payload. odata.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 odata.include-annotations preference in the Prefer header to request additional control information. Services supporting this MUST NOT omit control information required by the chosen odata.metadata parameter, and services MUST NOT exclude the odata.nextLink, odata.deltaLink, and odata.count if they are required by the response type.

3.1.1 odata.metadata=minimal

The odata.metadata=minimal format parameter indicates that the service SHOULD remove computable control information from the payload wherever possible. This is the default value for the odata.metadata parameter and will be assumed if no other value is specified in the Accept header or $format query option. The response payload MUST contain at least the following common annotations:

- odata.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
- odata.metadata: the ETag of the metadata document as applicable
- odata.etag: the ETag of the entity, as appropriate
**odata.count**: the total count of a collection of entities or collection of entity references, if requested

**odata.nextLink**: the next link of a collection with partial results

**odata.deltaLink**: the delta link for obtaining changes to the result, if requested

In addition, **odata** annotations MUST appear in the payload for cases where actual values are not the same as the computed values and MAY appear otherwise. When **odata** annotations appear in the payload, they are treated as exceptions to the computed values.

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

- **odata.mediaEtag**: the ETag of the stream, as appropriate
- **odata.mediaContentType**: the content type of the stream

### 3.1.2 odata.metadata=full

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

The full list of annotations that may appear in an **odata.metadata=full** response is as follows:

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

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

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

### 3.1.3 odata.metadata=none

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

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

---

*odata-json-format-v4.0-errata03-os-redlined*  
Standards Track Work Product  
Copyright © OASIS Open 2016. All Rights Reserved.  
02 June 2016  
Page 11 of 48
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 odata.count, if requested) as strings. This is in conformance with [I-JSON].

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 [ECMAScript] (see 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 payloads that format Edm.Int64 and Edm.Decimal values as strings MUST specify this format parameter in the media type returned in the Content-Type header.

The ExponentialDecimals=true format parameter indicates that the service MAY serialize Edm.Decimal numbers (including the odata.count, if requested) 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.
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 odata.metadata parameter to specify the amount of metadata included in the response.

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

Requests and responses MAY add the odata.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 odata.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 odata.type annotation, relative URLs are relative to the base type URL, which is

- the odata.type of the enclosing object, if one exists, otherwise
- the odata.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).

Example 2:

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

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

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 odata.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 odata.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 odata.streaming=true parameter.

Example 3: a payload with

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

can be assumed to support streaming, whereas a payload with

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

cannot be assumed to support streaming.

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

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

If present, the odata.context annotation MUST be the first property in the JSON object.

The odata.type annotation, if present, MUST appear next in the JSON object.

The odata.id and odata.etag annotations MUST appear before any property or property annotation.

All annotations for a structural or navigation property MUST appear as a group immediately before the property they annotate. The one exception is the odata.nextLink annotation of an expanded collection which MAY appear after the navigation property it annotates.

All other odata annotations can appear anywhere in the payload as long as they do not violate any of the above rules.

Annotations for navigation properties MUST appear after all structural properties.
4.5 Control Information

In addition to the “pure data” a message body MAY contain control information that is represented as annotations whose names start with odata followed by a dot.

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, including the odata namespace, MUST NOT stop processing and MUST NOT signal an error.

4.5.1 Annotation odata.context

The odata.context annotation returns the context URL (see [OData-Protocol]) for the payload. This URL can be absolute or relative.

The odata.context annotation is not returned if odata.metadata=none is requested. Otherwise it MUST be the first property of any JSON response.

The odata.context annotation 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:

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

4.5.2 Annotation odata.metadataEtag

The odata.metadataEtag annotation 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 odata.metadataEtag annotation to the metadata document's ETag in all responses when using odata.metadata=minimal or odata.metadata=full. If no ETag is returned when requesting the metadata document, then the service SHOULD NOT set the odata.metadataEtag annotation in any responses.

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

4.5.3 Annotation odata.type

The odata.type annotation 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, specified as a URI fragment. For all other types, the URI may be absolute or relative to the odata.type of the containing object. The root odata.type may be absolute or relative to the root context URL.

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-CSDL].

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

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

The following heuristics are used to determine the primitive type of a dynamic property in the absence of the odata.type annotation:

Boolean values have a first-class representation in JSON and do not need any additional annotations.

Numeric values have a first-class representation in JSON but are not further distinguished, so they include an odata.type annotation unless their type is Double.

The special floating-point values NaN, INF, and -INF are serialized as strings and MUST have an odata.type annotation to specify the numeric type.

String values do have a first class representation in JSON, but there is an obvious collision: OData also encodes a number of other primitive types as strings, e.g. DateTimeOffset, Int64 in the presence of the IEEE754Compatible format parameter etc. If a property appears in JSON string format, it should be treated as a string value unless the property is known (from the metadata document) to have a different type.

For more information on namespace- and alias-qualified names, see [OData-CSDL].

Example 5: entity of type Model.VipCustomer defined in the metadata document of the same service with a dynamic property of type Edm.Double and a value of positive infinity

```json
{
  "@odata.context": "http://host/service/$metadata#Customers/$entity",
  "@odata.type": "#Model.VipCustomer",
  "ID": 2,
  "DynamicLimit": "INF",
  "DynamicLimit@odata.type": "#Double",
  ...
}
```

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

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

### 4.5.4 Annotation odata.count

The odata.count annotation occurs only in responses and can annotate any collection of entities or collection of entity references, 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 Annotation odata.nextLink

The odata.nextLink annotation indicates that a response is only a subset of the requested collection of entities or collection of entity references. It contains a URL that allows retrieving the next subset of the requested collection.

This annotation can also be applied to expanded to-many navigation properties.

### 4.5.6 Annotation odata.deltaLink

The odata.deltaLink annotation contains a URL that can be used to retrieve changes to the current set of results. The odata.deltaLink annotation MUST only appear on the last page of results. A page of results MUST NOT have both an odata.deltaLink annotation and an odata.nextLink annotation.
4.5.7 Annotation odata.id

The *odata.id* annotation 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 *odata.id* annotation MUST appear in responses if *odata.metadata=full* is requested, or if *odata.metadata=minimal* is requested and 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 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 *odata.id* MUST be included if it does not match convention for the localized key values. If the *odata.id* is represented, it MAY be a relative URL.

If the entity is transient (i.e. cannot be read or updated), the *odata.id* annotation MUST appear and have the null value.

The *odata.id* annotation MUST NOT appear for a collection. Its meaning in this context is reserved for future versions of this specification.

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

4.5.8 Annotation odata.editLink and odata.readLink

The *odata.editLink* annotation contains the edit URL of the entity; see [OData-Protocol].

The *odata.readLink* annotation contains the read URL of the entity or collection; see [OData-Protocol].

The *odata.editLink* and *odata.readLink* annotations are ignored in request payloads and not written in responses if *odata.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 *odata.editLink* nor the *odata.readLink* annotation is present in an entity, the client uses this default value for the edit URL.

For updatable entities:

- The *odata.editLink* annotation is written if *odata.metadata=full* is requested or if *odata.metadata=minimal* is requested and the edit URL differs from the default value of the edit URL.
- The *odata.readLink* annotation is written if the read URL is different from the edit URL. If no *odata.readLink* annotation is present, the read URL is identical to the edit URL.

For read-only entities:

- The *odata.readLink* annotation is written if *odata.metadata=full* is requested or if *odata.metadata=minimal* is requested and its value differs from the default value of the read URL.
- The *odata.readLink* annotation may also be written if *odata.metadata=minimal* is specified in order to signal that an individual entity is read-only.

For collections:

- The *odata.readLink* annotation, if written, MUST be the request URL that produced the collection.
- The *odata.editLink* annotation MUST NOT be written as its meaning in this context is reserved for future versions of this specification.
4.5.9 Annotation odata.etag

The odata.etag annotation MAY be applied to an entity in a response. The value of the annotation 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 has changed.

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

The odata.etag annotation is ignored in request payloads and not written in responses if odata.metadata=none is requested.

4.5.10 Annotation odata.navigationLink and odata.associationLink

The odata.navigationLink annotation 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 odata.navigationLink annotation if odata.metadata=minimal has been specified on the request and the navigation link matches this computed value.

The odata.associationLink annotation 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 odata.associationLink annotation if the association link matches this computed value.

The odata.navigationLink and odata.associationLink annotations are ignored in request payloads and not written in responses if odata.metadata=none is requested.

4.5.11 Annotation odata.media*

For media entities and stream properties at least one of the annotations odata.mediaEditLink and odata.mediaReadLink MUST be included in responses if they don't follow standard URL conventions as defined in OData-URL or if odata.metadata=full is requested.

The odata.mediaEditLink annotation 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 media entities if it differs from the value of the odata.id, and for updatable stream properties if it differs from standard URL conventions.

The odata.mediaReadLink annotation 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 odata.mediaEditLink, if present, or the value of the odata.id for media entities if the associated odata.mediaEditLink is not present.

The odata.mediaContentType annotation MAY be included; its value SHOULD match the content type of the binary stream represented by the odata.mediaReadLink URL. This is only a hint; the actual content type will be included in a header when the resource is requested.

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

The odata.media* annotations are ignored in request payloads and not written in responses if odata.metadata=none is requested.

Example 7:

```json
{
    "@odata.context": "http://host/service/$metadata#Employees/$entity",
    "@odata.mediaReadLink": "Employees(1)/$value",
    "@odata.mediaContentType": "image/jpeg",
}```
"EmployeeID": 1,
...
}
5 Service Document

A service document in JSON is represented as a single JSON object with at least two properties; 
`odata.context` and `value`.

The value of the `odata.context` property 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-CSDL].

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 8:

```json
{
    "@odata.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": "ContosoMainSupplier",
            "title": "Contoso Ltd." Main Supplier",
            "kind": "Singleton"
        }
    ]
}
```
"url": "ContosoMainSupplier",
},
{
"name": "Human Resources",
"kind": "ServiceDocument",
"url": "http://host/HR/
}
]
6 Entity

An entity is serialized as a JSON object. 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 [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 9: entity with odata.metadata=minimal

```
{
  "@odata.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 10: entity with odata.metadata=full

```
{
  "@odata.context": "http://host/service/$metadata#Customers/$entity",
  "@odata.id": "Customers('ALFKI')",
  "@odata.etag": "W/\"MjAxMy0wNS0yN1QxMTo1OFo=\"",
  "@odata.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@odata.associationLink": "Customers('ALFKI')/Address/Country/$ref",
    "Country@odata.navigationLink": "Customers('ALFKI')/Address/Country"
  },
  "Orders@odata.associationLink": "Customers('ALFKI')/Orders/$ref",
  "Orders@odata.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; the 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.

7.1 Primitive Value

Primitive values are represented following the rules of [RFC7159].

Null values are represented as the JSON literal null.

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


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].

Values of type enumValue Enumeration values are represented as JSON strings using whose content satisfies the rule enumValue in [OData-ABNF]. The preferred representation is the enumerationMember, defined in [OData-ABNF], where 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 [GeoJSON], with the following modifications:

Keys SHOULD be ordered with type first, then coordinates, then any other keys.

The coordinates member of a LineString can have zero or more positions.

If the optional CRS object is present, it MUST be of type name, where the value of the name member of the contained properties object is an EPSG SRID legacy identifier.

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 odata.type annotation.

Example 11:

```json
{
  "NullValue": null,
  "TrueValue": true,
  "FalseValue": false,
  "BinaryValue": "T0RhdGE",
  "IntegerValue": -128,
  "DoubleValue": 3.1415926535897931,
  "StringValue": "Say \"Hello\", \n then go",
  "DateValue": "2012-12-03",
  "DateTimeOffsetValue": "2012-12-03T07:16:23Z",
  "DurationValue": "P12DT23H59M59.999999999999999S",
  "timeOfDayValue": "07:59:59.999",
  "GuidValue": "01234567-89ab-cdef-0123-456789abcdef",
  "Int64Value": 0,
  "DecimalValue": 34.95,
  "NameValue": "Say \"Hello\", \n then go",
}
"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, including odata annotations.

Example 12:

```
{
  "@odata.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 13: partial collection of strings with next link

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

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 14: partial collection of complex values with next link

```
{
  "PhoneNumbers": [
    {
      "Number": "425-555-1212",
      "Type": "Home"
    },
    {
      "@odata.type": "#Model.CellPhoneNumber",
      "Number": "425-555-0178"
    }
  ],
  "PhoneNumbers@odata.nextLink": "..."
}
```
"Type": "Cell",
   "Carrier": "Sprint"
  }
 },
 "PhoneNumbers@odata.nextLink": "...
}
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 name/value pair. The name is the name of the property, followed by @odata.navigationLink. The 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 odata.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 15:

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

8.2 Association Link

The association link for a navigation property is represented as a name/value pair. The name is the name of the property, followed by @odata.associationLink. The 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 odata.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 16:

```json
{
  "@odata.context": "http://host/service/$metadata#Customers/$entity",
  ...
  "Orders@odata.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 be annotated with odata.context, odata.count or odata.nextLink. If a navigation property is expanded with the suffix /$count, only the odata.count annotation is represented.
Example 17:

```json
{
   "@odata.context": "http://host/service/$metadata#Customers/$entity",
   ...
   "Orders@odata.count": 42,
   "Orders": [ ... ],
   "Orders@odata.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 18:** inserting a new order for a new customer with order items related to existing products:

```json
{
   "ID": 11643,
   "Amount": 100,
   ...
   "Customer": {
      "ID": "ANEWONE",
      ...
   },
   "Items": [
      {
         "Product@odata.bind": "Products(28)",
         "Quantity": 1,
         ...
      },
      {
         "Product@odata.bind": "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. A bind operation is encoded as a property annotation `odata.bind` on the navigation property it belongs to and has a single value for singleton 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.**

The values are the ids of the related entities. They MAY be absolute or relative URLs.

For insert operations collection navigation property bind operations and deep insert operations can be combined. In this case, 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.

**Example 19:** assign an existing product to an existing category with a partial update request

```
PATCH http://host/service/Products(42) HTTP/1.1
```
{ "Category@odata.bind": "Categories(6)" }
9 Stream Property

An entity or complex type instance can have one or more stream properties. The actual stream data is not contained in the representation. Instead stream property data is read and edited via URLs. The value for a stream property contains the URLs for reading and editing the stream data along with other metadata for the stream.

The value of a stream property is represented as a set of odata.media* annotations.

Example 20:

```json
{
   "@odata.context": "http://host/service/$metadata#Products/$entity",
   ...
   "Thumbnail@odata.mediaReadLink": "http://server/Thumbnail546.jpg",
   "Thumbnail@odata.mediaEditLink": "http://server/uploads/Thumbnail546.jpg",
   "Thumbnail@odata.mediaContentType": "image/jpeg",
   "Thumbnail@odata.mediaEtag": "W/"####",
   ...
}
```
10 Media Entity

Media entities are entities that describe a media resource, for example a photo. They are represented as entities that contain additional `odata.media*` annotations.

Example 21:

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

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

Example 23: collection of primitive values

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

Example 24: empty collection of primitive values

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

Example 25: complex value

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

Example 26: empty collection of complex values

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

Note: the context URL is optional in requests.
12 Collection of Entities

A collection of entities is represented as a JSON object containing a name/value pair named `value`. It MAY contain `odata.context`, `odata.count`, `odata.nextLink`, or `odata.deltaLink` annotations. If present, the `odata.context` annotation MUST be the first name/value pair in the response.

The `odata.count` name/value pair represents the number of entities in the collection. If present and the `odata.streaming=true` content type parameter is set, it MUST come before the `value` name/value pair. If the response represents a partial result, the `odata.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 `odata.nextLink` annotation MUST be included in a response that represents a partial result.

Example 27:

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

An entity reference (see [OData-Protocol]) MAY take the place of an entity instance 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 odata.type.

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 MUST contain an odata.context annotation and MAY contain odata.count, odata.nextLink, or odata.deltaLink annotations.

Example 28: entity reference to order 10643

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

Example 29: collection of entity references

```json
{
    "@odata.context": "http://host/service/$metadata#Collection($ref)",
    "value": [
        "@odata.id": "Orders(10643)"
    ]
}
```
14 Delta Response

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

Responses from a delta request are returned as a JSON object. The JSON object MUST contain an array-valued property named value containing all added, changed, or deleted entities, as well as added 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 for retrieving subsequent changes once the current set of changes has been applied to the initial set.

If the response from the delta link contains an odata.count annotation, the returned number MUST include all added, changed, or deleted entities to be returned, as well as added or deleted links.

Example 30: 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

```json
{
  "@odata.context":"http://host/service/$metadata#Customers/$delta",
  "@odata.count":5,
  "value": [
    {
      "@odata.id":"Customers('BOTTM')",
      "ContactName":"Susan Halvenstern"
    },
    {
      "@odata.context":"#Customers/$deletedLink",
      "source":"Customers('ALFKI')",
      "relationship":"Orders",
      "target":"Orders(10643)"
    },
    {
      "@odata.context":"#Customers/$link",
      "source":"Customers('BOTTM')",
      "relationship":"Orders",
      "target":"Orders(10645)"
    },
    {
      "@odata.context":"#Orders/$entity",
      "@odata.id":"Orders(10643)",
      "ShippingAddress":{
        "Street":"23 Tsawassen Blvd.",
        "City":"Tsawassen",
        "Region":"BC",
        "PostalCode":"T2F 8M4"
      },
    },
    {
      "@odata.context":"#Customers/$deletedEntity",
      "id":"Customers('ANTON')",
      "reason":"deleted"
    }
  ]
}
```
14.1 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 principle 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 odata.context annotation to specify the entity set of the entity, regardless of the specified odata.metadata value.

Entities include annotations for selected navigation links based on odata.metadata but MUST NOT include expanded navigation properties inline.

14.2 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 deleted-entity object MUST include the following properties, regardless of the specified odata.metadata value:

- odata.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 odata.id returned or computed when calling GET on resource), which may be absolute or relative

The deleted-entity object MAY include the following optional property, regardless of the specified odata.metadata value:

- 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).

14.3 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 odata.metadata value:

- odata.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 name of the relationship navigation property on the parent source object

target – The id of the related entity, which may be absolute or relative

14.4 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 odata.metadata value

odata.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 name of the relationship navigation property on the parent source object

target – The id of the related entity, which may be absolute or relative

Note: target is only necessary for multi-valued navigation properties. For single-valued navigation properties source and relationship are sufficient to describe the change. Clients should not require a value for target for single-valued navigation properties as it will become optional in future versions of this specification.
15Bound Function

A function that is bound to the current entity 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-CSDL].

Functions that are bound to a collection of entities are advertised in representations of that collection.

A function may have multiple overloads with different parameters. If function overloads exist that cannot be bound to the current entity type, the name SHOULD address a specific function overload by appending the parentheses-enclosed, comma-separated list of non-binding parameter names, see rule qualifiedFunctionName in [OData-ABNF].

If odata.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 bound function or action URL. If the URL in the target name/value pair cannot be used to invoke all overloads for the function, then the function name MUST be distinguished by appending the parentheses-enclosed, comma-separated list of non-binding parameter names.

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

If odata.metadata=minimal is requested, the target name/value pair MUST be included if its value differs from the canonical function or action URL.

Example 31: minimal representation of a function where all overloads are applicable

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

Example 32: full representation of a specific overload

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

Example 33: full representation in a collection

```
{  
  "@odata.context": "http://host/service/$metadata#Employees",  
  "#Model.RemainingVacation": {  
    "title": "Remaining Vacation",  
    "target": "Managers(22)/Employees/RemainingVacation"
  },  
  "value": [ ... ]
}
```
### 16 Bound Action

An action that is bound to the current entity 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-CSDL].

Actions that are bound to a collection of entities are advertised in representations of that collection.

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

**Example 34: minimal representation in an entity**

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

**Example 35: full representation in an entity:**

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

**Example 36: full representation in a collection**

```json
{
  "@odata.context": "http://host/service/$metadata#LeaveRequests",
  "#Model.Approval": {
    "title": "Approve All Leave Requests",
    "target": "Managers(22)/Inbox/Approval"
  },
  "value": [ ... ]
}
```
17 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.

Any parameter values not specified in the JSON object are assumed to have the null value.

Example 37:

```json
{
    "param1": 42,
    "param2": {
        "Street": "One Microsoft Way",
        "Zip": 98052
    },
    "param3": [ 1, 42, 99 ],
    "param4": null
}
```
18 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 used to include control information in many payloads.

Annotations are name/value pairs that have a dot (.) as part of the name. All annotations that start with odata are 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-CSDL].

The namespace or alias odata is reserved for future extensions of the protocol and format. Custom annotations are annotations that have a non-empty prefix 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 (see [OData-CSDL]).

Example 38:

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

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 they are placed next to the annotated model construct.

Example 38:

```json
{
    "@odata.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 }
        }
    ]
}
```

18.1 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 namespace- or alias-qualified name of the annotation, i.e. the namespace or alias of the schema that defines the term, followed by a dot (.).
followed by the name of the term. The namespace or alias MUST be defined in the metadata document, see [OData-CSDL], annotation identifier.
The value MUST be an appropriate value for the annotation.

18.2 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 placed next to the annotated name/value pair and represented as a single name/value pair.

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 namespace- or alias-qualified name of the annotation, followed by a dot (.), followed by the name of the term. The namespace or alias MUST be defined in the metadata document, see [OData-CSDL], identifier.
The value MUST be an appropriate value for the annotation.
19 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 a JSON object.

This 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 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.

The value for the `message` name/value pair MUST be a human-readable, language-dependent representation of 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.

The value for the `target` name/value pair is the target of the particular error (for example, the name of the property in error).

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. The `innererror` name/value pair SHOULD only be used in development environments in order to guard against potential security concerns around information disclosure.

Error responses MAY contain annotations in any of its JSON objects.

Example 39:

```json
{
  "error": {
    "code": "501",
    "message": "Unsupported functionality",
    "target": "query",
    "details": [
      {
        "code": "301",
        "target": "$search",
        "message": "$search query option not supported"
      }
    ],
    "innererror": {
      "trace": [...],
      "context": {...}
    }
  }
}
```
20 Extensibility

Implementations can add custom 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.
21 Security 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 [RFC7159] as starting point.
22 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:
   a. understand odata.metadata=minimal (section 3.1.1) or
   b. explicitly specify odata.metadata=none (section 3.1.3) or odata.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)
   a. defined in this specification (client)
   b. exposed by the service (service)
4. MUST interpret all odata annotations defined according to the OData-Version header of the payload (section 4.5)
5. MUST be prepared to receive any annotations, including custom annotations and odata annotations not defined in the OData-Version header of the payload (section 20)
6. MUST NOT require odata.streaming=true in the Content-Type header (section 4.4)

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

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

The contributions of the OASIS OData Technical Committee members, enumerated in [OData-Protocol], are gratefully acknowledged.
## Appendix B. Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Editor</th>
<th>Changes Made</th>
</tr>
</thead>
<tbody>
<tr>
<td>Working Draft 01</td>
<td>2012-08-22</td>
<td>Michael Pizzo</td>
<td>Translated Contribution to OASIS format/template</td>
</tr>
<tr>
<td>Working Draft 01.1</td>
<td>2013-01-31</td>
<td>Ralf Handl</td>
<td>Adopted new, more concise JSON format</td>
</tr>
</tbody>
</table>
| Committee Specification Draft 01| 2013-04-26 | Ralf Handl, Michael Pizzo | Expanded error information  
Added enumerations  
Fleshed out descriptions and examples and addressed numerous editorial and technical issues through processed through the TC  
Added Conformance section                                                   |
| Committee Specification Draft 02| 2013-07-01 | Ralf Handl, Michael Pizzo | Improved rules for odata.id, odata.editLink, and odata.readLink  
Improved action/function advertisement  
Improved entity references  
Improved rules for relative URLs  
Simplified delta responses  
GeoJSON for Geo types  
Improved description of primitive value representation  
Improved examples, aligned with Atom format specification  
Aligned terms across specifications |
| Committee Specification 01      | 2013-07-30 | Ralf Handl, Michael Pizzo | Non-Material Changes                                                                                                                             |
| Committee Specification Draft 03| 2013-10-03 | Ralf Handl, Michael Pizzo | Annotations start with @  
Next link for collections of primitive and complex type  
Null values in collections of primitive and complex type  
Improved description of relative URL resolution                                      |
<p>| Committee Specification 02      | 2013-11-04 | Michael Pizzo, Ralf Handl, Martin Zurmuehl | Non-Material Changes                                                                                                                             |
| OASIS Specification             | 2014-02-24 | Michael Pizzo, Ralf Handl, Martin Zurmuehl | Non-Material Changes                                                                                                                             |
| Errata 01                       | 2014-07-24 | Michael Pizzo, Ralf Handl, | Minor changes and improvements                                                                                                                 |</p>
<table>
<thead>
<tr>
<th>Errata 02</th>
<th>2014-10-29</th>
<th>Michael Pizzo, Ralf Handl, Martin Zumuehl</th>
<th>Repaired mechanical error in the editable source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Errata 03</td>
<td>2016-03-03</td>
<td>Michael Pizzo, Ralf Handl, Martin Zumuehl</td>
<td>Minor changes and clarifications</td>
</tr>
</tbody>
</table>