OData Version 4.0 Part 1: Protocol Plus Errata 01

OASIS Standard incorporating Approved Errata 01

04 September 2014

Specification URIs

This version:

http://docs.oasis-open.org/odata/odata/v4.0/errata01/os/complete/part1-protocol/odata-v4.0-errata01-os-part1-protocol-complete.doc (Authoritative)
http://docs.oasis-open.org/odata/odata/v4.0/errata01/os/complete/part1-protocol/odata-v4.0-errata01-os-part1-protocol-complete.html
http://docs.oasis-open.org/odata/odata/v4.0/errata01/os/complete/part1-protocol/odata-v4.0-errata01-os-part1-protocol-complete.pdf

Previous version:

http://docs.oasis-open.org/odata/odata/v4.0/os/part1-protocol/odata-v4.0-os-part1-protocol.doc (Authoritative)
http://docs.oasis-open.org/odata/odata/v4.0/os/part1-protocol/odata-v4.0-os-part1-protocol.html
http://docs.oasis-open.org/odata/odata/v4.0/os/part1-protocol/odata-v4.0-os-part1-protocol.pdf

Latest version:

http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part1-protocol.doc (Authoritative)
http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part1-protocol.html
http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part1-protocol.pdf

Technical Committee:

OASIS Open Data Protocol (OData) TC

Chairs:

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

Editors:

Michael Pizzo (mikep@microsoft.com), Microsoft
Ralf Handl (ralf.handl@sap.com), SAP AG
Martin Zurmuehl (martin.zurmuehl@sap.com), SAP AG

Additional artifacts:

This prose specification is one component of a Work Product that also includes:

- List of Errata items. *OData Version 4.0 Errata 01*. Edited by Michael Pizzo, Ralf Handl, Martin Zurmuehl, and Hubert Heijkers. 04 September 2014. OASIS Approved Errata. http://docs.oasis-open.org/odata/odata/v4.0/errata01/os/odata-v4.0-errata01-os.html.
- *OData Version 4.0 Part 1: Protocol Plus Errata 01* (this document). Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. 04 September 2014. OASIS Standard incorporating Approved Errata 01. http://docs.oasis-open.org/odata/odata/v4.0/errata01/os/complete/part1-protocol/odata-v4.0-errata01-os-part1-protocol-complete.html.
- *OData Version 4.0 Part 2: URL Conventions Plus Errata 01*. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. 04 September 2014. OASIS Standard incorporating Approved Errata 01. http://docs.oasis-open.org/odata/odata/v4.0/errata01/os/complete/part2-url-conventions/odata-v4.0-errata01-os-part2-url-conventions-complete.html.

 Copyright © OASIS Open 2014. All Rights Reserved.

- *OData Version 4.0 Part 3: Common Schema Definition Language (CSDL) Plus Errata 01*. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. 04 September 2014. OASIS Standard incorporating Approved Errata 01. http://docs.oasis-open.org/odata/odata/v4.0/errata01/os/complete/part3-csdl/odata-v4.0-errata01-os-part3-csdl-complete.html.
- ABNF components: OData ABNF Construction Rules Version 4.0 and OData ABNF Test Cases. http://docs.oasis-open.org/odata/odata/v4.0/errata01/os/complete/abnf/.
- Vocabulary components: OData Core Vocabulary, OData Measures Vocabulary and OData Capabilities Vocabulary. http://docs.oasis-open.org/odata/odata/v4.0/errata01/os/complete/vocabularies/.
- XML schemas: OData EDMX XML Schema and OData EDM XML Schema. http://docs.oasis-open.org/odata/odata/v4.0/errata01/os/complete/schemas/.
- OData Metadata Service Entity Model: http://docs.oasis-open.org/odata/odata/v4.0/errata01/os/complete/models/.
- Change-marked (redlined) versions of *OData Version 4.0 Part 1, Part 2,* and *Part 3*. OASIS Standard incorporating Draft 01 of Errata 01. http://docs.oasis-open.org/odata/odata/v4.0/errata01/os/redlined/.

Related work:

This specification is related to:

- *OData Version 4.0 Part 1: Protocol*. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. 24 February 2014. OASIS Standard. http://docs.oasis-open.org/odata/odata/v4.0/os/part1-protocol/odata-v4.0-os-part1-protocol.html.
- *OData Atom Format Version 4.0*. Edited by Martin Zurmuehl, Michael Pizzo, and Ralf Handl. Latest version. http://docs.oasis-open.org/odata/odata-atom-format/v4.0/odata-atom-format-v4.0.html.
- *OData JSON Format Version 4.0*. Edited by Ralf Handl, Michael Pizzo, and Mark Biamonte. Latest version. http://docs.oasis-open.org/odata/odata-json-format/v4.0/odata-json-format-v4.0.html.

Declared XML namespaces:

- http://docs.oasis-open.org/odata/ns/edmx
- http://docs.oasis-open.org/odata/ns/edm

Abstract:

The Open Data Protocol (OData) enables the creation of REST-based data services, which allow resources, identified using Uniform Resource Locators (URLs) and defined in an Entity Data Model (EDM), to be published and edited by Web clients using simple HTTP messages. This document defines the core semantics and facilities of the protocol.

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 Technical Committee 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-Part1]

 Copyright © OASIS Open 2014. All Rights Reserved.

OData Version 4.0 Part 1: Protocol Plus Errata 01. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. 04 September 2014. OASIS Standard incorporating Approved Errata 01. http://docs.oasis-open.org/odata/odata/v4.0/errata01/os/complete/part1-protocol/odata-v4.0-errata01-os-part1-protocol-complete.html. Latest version: http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part1-protocol.html.

 Copyright © OASIS Open 2014. All Rights Reserved.

Notices

Copyright © OASIS Open 2014. 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 10
1.1 Terminology 10
1.2 Normative References 10
1.3 Typographical Conventions 11
2 Overview 12
3 Data Model 13
3.1 Annotations 13
4 Service Model 15
4.1 Entity-Ids and Entity References 15
4.2 Read URLs and Edit URLs 15
4.3 Transient Entities 15
5 Versioning 16
5.1 Protocol Versioning 16
5.2 Model Versioning 16
6 Extensibility 17
6.1 Query Option Extensibility 17
6.2 Payload Extensibility 17
6.3 Action/Function Extensibility 17
6.4 Vocabulary Extensibility 17
6.5 Header Field Extensibility 18
6.6 Format Extensibility 18
7 Formats 19
8 Header Fields 20
8.1 Common Headers 20
8.1.1 Header `Content-Type` 20
8.1.2 Header `Content-Encoding` 20
8.1.3 Header `Content-Language` 20
8.1.4 Header `Content-Length` 20
8.1.5 Header `OData-Version` 20
8.2 Request Headers 21
8.2.1 Header `Accept` 21
8.2.2 Header `Accept-Charset` 21
8.2.3 Header `Accept-Language` 21
8.2.4 Header `If-Match` 21
8.2.5 Header `If-None-Match` 21
8.2.6 Header `OData-Isolation` 22
8.2.7 Header `OData-MaxVersion` 22
8.2.8 Header `Prefer` 22
8.2.8.1 Preference `odata.allow-entityreferences` 23
8.2.8.2 Preference `odata.callback` 23
8.2.8.3 Preference `odata.continue-on-error` 24
8.2.8.4 Preference `odata.include-annotations` 24

Copyright © OASIS Open 2014. All Rights Reserved.

8.2.8.5 Preference odata.maxpagesize 25
8.2.8.6 Preference odata.track-changes 25
8.2.8.7 Preference return=representation and return=minimal 25
8.2.8.8 Preference respond-async 26
8.2.8.9 Preference wait 26
8.3 Response Headers 26
8.3.1 Header ETag 26
8.3.2 Header Location 27
8.3.3 Header OData-EntityId 27
8.3.4 Header Preference-Applied 27
8.3.5 Header Retry-After 27
9 Common Response Status Codes 28
9.1 Success Responses 28
9.1.1 Response Code 200 OK 28
9.1.2 Response Code 201 Created 28
9.1.3 Response Code 202 Accepted 28
9.1.4 Response Code 204 No Content 28
9.1.5 Response Code 3xx Redirection 28
9.1.6 Response Code 304 Not Modified 28
9.2 Client Error Responses 29
9.2.1 Response Code 404 Not Found 29
9.2.2 Response Code 405 Method Not Allowed 29
9.2.3 Response Code 410 Gone 29
9.2.4 Response Code 412 Precondition Failed 29
9.3 Server Error Responses 29
9.3.1 Response Code 501 Not Implemented 29
9.4 In-Stream Errors 29
10 Context URL 30
10.1 Service Document 30
10.2 Collection of Entities 30
10.3 Entity 31
10.4 Singleton 31
10.5 Collection of Derived Entities 31
10.6 Derived Entity 32
10.7 Collection of Projected Entities 32
10.8 Projected Entity 32
10.9 Collection of Projected Expanded Entities 33
10.10 Projected Expanded Entity 33
10.11 Collection of Entity References 33
10.12 Entity Reference 34
10.13 Property Value 34
10.14 Collection of Complex or Primitive Types 34
10.15 Complex or Primitive Type 34
10.16 Operation Result 34

 Copyright © OASIS Open 2014. All Rights Reserved.

10.17 Delta Response 35
10.18 Item in a Delta Response 35
10.19 $all Response 35
10.20 $crossjoin Response 35
11 Data Service Requests 36
11.1 Metadata Requests 36
11.1.1 Service Document Request 36
11.1.2 Metadata Document Request 36
11.1.3 Metadata Service Document Request 36
11.2 Requesting Data 36
11.2.1 Evaluating System Query Options 37
11.2.2 Requesting Individual Entities 37
11.2.3 Requesting Individual Properties 37
11.2.3.1 Requesting a Property's Raw Value using $value 38
11.2.4 Specifying Properties to Return 38
11.2.4.1 System Query Option $select 38
11.2.4.2 System Query Option $expand 39
11.2.4.2.1 Expand Options 39
11.2.4.2.1.1 Expand Option $levels 40
11.2.5 Querying Collections 40
11.2.5.1 System Query Option $filter 40
11.2.5.1.1 Built-in Filter Operations 40
11.2.5.1.2 Built-in Query Functions 41
11.2.5.1.3 Parameter Aliases 43
11.2.5.2 System Query Option $orderby 43
11.2.5.3 System Query Option $top 44
11.2.5.4 System Query Option $skip 44
11.2.5.5 System Query Option $count 44
11.2.5.6 System Query Option $search 45
11.2.5.7 Server-Driven Paging 45
11.2.6 Requesting Related Entities 46
11.2.7 Requesting Entity References 46
11.2.8 Resolving an Entity-Id 46
11.2.9 Requesting the Number of Items in a Collection 47
11.2.10 System Query Option $format 47
11.3 Requesting Changes 48
11.3.1 Delta Links 48
11.3.2 Using Delta Links 48
11.4 Data Modification 49
11.4.1 Common Data Modification Semantics 49
11.4.1.1 Use of ETags for Avoiding Update Conflicts 49
11.4.1.2 Handling of DateTimeOffset Values 49
11.4.1.3 Handling of Properties Not Advertised in Metadata 49
11.4.1.4 Handling of Consistency Constraints 49
11.4.1.5 Returning Results from Data Modification Requests 50
11.4.2 Create an Entity 50

 Copyright © OASIS Open 2014. All Rights Reserved.

11.4.2.1 Link to Related Entities When Creating an Entity 50
11.4.2.2 Create Related Entities When Creating an Entity 51
11.4.3 Update an Entity 51
11.4.4 Upsert an Entity 52
11.4.5 Delete an Entity 52
11.4.6 Modifying Relationships between Entities 53
11.4.6.1 Add a Reference to a Collection-Valued Navigation Property 53
11.4.6.2 Remove a Reference to an Entity 53
11.4.6.3 Change the Reference in a Single-Valued Navigation Property 53
11.4.7 Managing Media Entities 53
11.4.7.1 Creating a Media Entity 53
11.4.7.2 Editing a Media Entity Stream 54
11.4.7.3 Deleting a Media Entity 54
11.4.8 Managing Stream Properties 54
11.4.8.1 Editing Stream Values 54
11.4.8.2 Deleting Stream Values 54
11.4.9 Managing Values and Properties Directly 54
11.4.9.1 Update a Primitive Property 54
11.4.9.2 Set a Value to Null 55
11.4.9.3 Update a Complex Property 55
11.4.9.4 Update a Collection Property 55
11.5 Operations 55
11.5.1 Binding an Operation to a Resource 55
11.5.2 Advertising Available Operations within a Payload 56
11.5.3 Functions 56
11.5.3.1 Invoking a Function 56
11.5.3.1.1 Inline Parameter Syntax 57
11.5.3.2 Function overload resolution 57
11.5.4 Actions 58
11.5.4.1 Invoking an Action 58
11.5.4.2 Action Overload Resolution 58
11.6 Asynchronous Requests 59
11.7 Batch Requests 59
11.7.1 Batch Request Headers 59
11.7.2 Batch Request Body 60
11.7.3 Change Sets 62
11.7.3.1 Referencing New Entities in a Change Set 62
11.7.4 Responding to a Batch Request 63
11.7.5 Asynchronous Batch Requests 65
12 Security Considerations 67
12.1 Authentication 67
13 Conformance 68
13.1 OData Service Conformance Levels 68
13.1.1 OData Minimal Conformance Level 68
13.1.2 OData Intermediate Conformance Level 69
13.1.3 OData Advanced Conformance Level 70
13.2 Interoperable OData Clients 70

 Copyright © OASIS Open 2014. All Rights Reserved.

Appendix A. Acknowledgments 71
Appendix B. Revision History 72

 Copyright © OASIS Open 2014. All Rights Reserved.

1 Introduction

The Open Data Protocol (OData) enables the creation of REST-based data services, which allow resources, identified using Uniform Resource Locators (URLs) and defined in a data model, to be published and edited by Web clients using simple HTTP messages. This specification defines the core semantics and the behavioral aspects of the protocol.

The **[OData-URL]** specification defines a set of rules for constructing URLs to identify the data and metadata exposed by an OData service as well as a set of reserved URL query options.

The **[OData-CSDL]** specification defines an XML representation of the entity data model exposed by an OData service.

The **[OData-Atom]** and **[OData-JSON]** documents specify the format of the resource representations that are exchanged using OData.

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

[OData-ABNF] *OData ABNF Construction Rules Version 4.0*.
See link in "Additional artifacts" section on cover page.

[OData-Atom] *OData ATOM Format Version 4.0*.
See link in "Related work" section on cover page.

[OData-CSDL] *OData Version 4.0 Part 3: Common Schema Definition Language (CSDL)*.
See link in "Additional artifacts" section on cover page.

[OData-JSON] *OData JSON Format Version 4.0.*
See link in "Related work" section on cover page.

[OData-URL] *OData Version 4.0 Part 2: URL Conventions*.
See link in "Additional artifacts" section on cover page.

[OData-VocCap] *OData Capabilities Vocabulary.*
See link in "Additional artifacts" section on cover page.

[OData-VocCore] *OData Core Vocabulary.*
See link in "Additional artifacts" section on cover page.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November, 1996. http://www.ietf.org/rfc/rfc2046.txt.

[RFC2119] Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels”, BCP 14, RFC 2119, March 1997. http://www.ietf.org/rfc/rfc2119.txt.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, “HTTP Authentication: Basic and Digest Access Authentication”, RFC 2617, June 1999. http://www.ietf.org/rfc/rfc2617.txt.

[RFC3987] Duerst, M. and, M. Suignard, “Internationalized Resource Identifiers (IRIs)”, RFC 3987, January 2005. http://www.ietf.org/rfc/rfc3987.txt.

[RFC5023] Gregorio, J., Ed., and B. de hOra, Ed., “The Atom Publishing Protocol.”, RFC 5023, October 2007. http://tools.ietf.org/html/rfc5023.

[RFC5789] Dusseault, L., and J. Snell, “Patch Method for HTTP”, RFC 5789, March 2010. http://tools.ietf.org/html/rfc5789.

Copyright © OASIS Open 2014. All Rights Reserved.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., “Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing”, RFC 7230, June 2014. http://www.ietf.org/rfc/rfc7230.txt.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., “Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content”, RFC 7231, June 2014. http://www.ietf.org/rfc/rfc7231.txt.

[RFC7232] Fielding, R., Ed. and J. Reschke, Ed., “Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests”, RFC 7232, June 2014. http://www.ietf.org/rfc/rfc7232.txt.

[RFC7240] Snell, J., "Prefer Header for HTTP", RFC 7240, June 2014. http://www.ietf.org/rfc/rfc7240.txt.

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.

 Copyright © OASIS Open 2014. All Rights Reserved.

2 Overview

The OData Protocol is an application-level protocol for interacting with data via RESTful interfaces. The protocol supports the description of data models and the editing and querying of data according to those models. It provides facilities for:

- Metadata: a machine-readable description of the data model exposed by a particular data provider.
- Data: sets of data entities and the relationships between them.
- Querying: requesting that the service perform a set of filtering and other transformations to its data, then return the results.
- Editing: creating, updating, and deleting data.
- Operations: invoking custom logic
- Vocabularies: attaching custom semantics

The OData Protocol is different from other REST-based web service approaches in that it provides a uniform way to describe both the data and the data model. This improves semantic interoperability between systems and allows an ecosystem to emerge.

Towards that end, the OData Protocol follows these design principles:

- Prefer mechanisms that work on a variety of data stores. In particular, do not assume a relational data model.
- Extensibility is important. Services should be able to support extended functionality without breaking clients unaware of those extensions.
- Follow REST principles.
- OData should build incrementally. A very basic, compliant service should be easy to build, with additional work necessary only to support additional capabilities.
- Keep it simple. Address the common cases and provide extensibility where necessary.

 Copyright © OASIS Open 2014. All Rights Reserved.

3 Data Model

This section provides a high-level description of the *Entity Data Model (EDM)*: the abstract data model that is used to describe the data exposed by an OData service. An OData Metadata Document is a representation of a service's data model exposed for client consumption.

The central concepts in the EDM are entities, relationships, entity sets, actions, and functions.

Entities are instances of entity types (e.g. `Customer`, `Employee`, etc.).

Entity types are named structured types with a key. They define the named properties and relationships of an entity. Entity types may derive by single inheritance from other entity types.

The *key* of an entity type is formed from a subset of the primitive properties (e.g. `CustomerId`, `OrderId`, `LineId`, etc.) of the entity type.

Complex types are keyless named structured types consisting of a set of properties. These are value types whose instances cannot be referenced outside of their containing entity. Complex types are commonly used as property values in an entity or as parameters to operations.

Properties declared as part of a structured type's definition are called *declared properties*. Instances of structured types may contain additional undeclared *dynamic properties*. A dynamic property cannot have the same name as a declared property. Entity or complex types which allow clients to persist additional undeclared properties are called *open types*.

Relationships from one entity to another are represented as *navigation properties.* Navigation properties are generally defined as part of an entity type, but can also appear on entity instances as undeclared *dynamic navigation properties*. Each relationship has a cardinality.

Enumeration types are named primitive types whose values are named constants with underlying integer values.

Type definitions are named primitive types with fixed facet values such as maximum length or precision. Type definitions can be used in place of primitive typed properties, for example, within property definitions.

Entity sets are named collections of entities (e.g. `Customers` is an entity set containing `Customer` entities). An entity's key uniquely identifies the entity within an entity set. If multiple entity sets use the same entity type, the same combination of key values can appear in more than one entity set and identifies different entities, one per entity set where this key combination appears. Each of these entities has a different entity-id. Entity sets provide entry points into the data model.

Operations allow the execution of custom logic on parts of a data model. *Functions* are operations that do not have side effects and may support further composition, for example, with additional filter operations, functions or an action. *Actions* are operations that allow side effects, such as data modification, and cannot be further composed in order to avoid non-deterministic behavior. Actions and functions are either *bound* to a type, enabling them to be called as members of an instance of that type, or unbound, in which case they are called as static operations. *Action imports* and *function imports* enable unbound actions and functions to be called from the service root.

Singletons are single entities which are accessed as children of the entity container.

An OData *resource* is anything in the model that can be addressed (an entity set, entity, property, or operation).

Refer to **[OData-CSDL]** for more information on the OData entity data model.

3.1 Annotations

Model and instance elements can be decorated with *Annotations*.

Annotations can be used to specify an individual fact about an element, such as whether it is read-only, or to define a common concept, such as a person or a movie.

 Copyright © OASIS Open 2014. All Rights Reserved.

Applied *annotations* consist of a *term* (the namespace-qualified name of the annotation being applied), a *target* (the model or instance element to which the term is applied), and a *value*. The value may be a static value, or an expression that may contain a path to one or more properties of an annotated entity.

Annotation terms are defined in metadata and have a name and a type.

A set of related terms in a common namespace comprises a *Vocabulary*.

 Copyright © OASIS Open 2014. All Rights Reserved.

4 Service Model

OData services are defined using a common data model. The service advertises its concrete data model in a machine-readable form, allowing generic clients to interact with the service in a well-defined way.

An OData service exposes two well-defined resources that describe its data model; a service document and a metadata document.

The *service document* lists entity sets, functions, and singletons that can be retrieved. Clients can use the service document to navigate the model in a hypermedia-driven fashion.

The *metadata document* describes the types, sets, functions and actions understood by the OData service. Clients can use the metadata document to understand how to query and interact with entities in the service.

In addition to these two “fixed” resources an OData service consists of dynamic resources. The URLs for many of these resources can be computed from the information in the metadata document.

See Requesting Data and Data Modification for details.

4.1 Entity-Ids and Entity References

Whereas entities within an entity set are uniquely identified by their key values, entities are also uniquely identified by a durable, opaque, globally unique *entity-id*. The entity-id MUST be an IRI as defined in **[RFC3987]** and MAY be expressed in payloads and URLs as a relative reference as appropriate. While the client MUST be prepared to accept any IRI, services MUST use valid URIs in this version of the specification since there is currently no lossless representation of an IRI in the `OData-EntityId` header.

Services are strongly encouraged to use the canonical URL for an entity as defined in **[OData-URL]** as its entity-id, but clients cannot assume the entity-id can be used to locate the entity unless the `Core.DereferenceableIDs` term is applied to the entity container, nor can the client assume any semantics from the structure of the entity-id. The canonical resource `$entity` provides a general mechanism for resolving an entity-id into an entity representation.

Services that use the standard URL conventions for entity-ids annotate their entity container with the term `Core.ConventionalIDs`, see **[OData-VocCore]**.

Entity references refer to an entity using the entity's entity-id.

4.2 Read URLs and Edit URLs

The read URL of an entity is the URL that can be used to read the entity.

The edit URL of an entity is the URL that can be used to update or delete the entity.

The edit URL of a property is the edit URL of the entity with appended segment(s) containing the path to the property.

Services are strongly encouraged to use the canonical URL for an entity as defined in **[OData-URL]** for both the read URL and the edit URL of an entity, with a cast segment to the type of the entity appended to the canonical URL if the type of the entity is derived from the declared type of the entity set. However, clients cannot assume this convention and must use the links specified in the payload according to the appropriate format as the two URLs may be different from one another, or one or both of them may differ from convention.

4.3 Transient Entities

Transient entities are instances of an entity type that are “calculated on the fly” and only exist within a single payload. They cannot be reread or updated and consequently possess neither a stable entity-id nor a read URL or an update URL.

 Copyright © OASIS Open 2014. All Rights Reserved.

5 Versioning

Versioning enables clients and services to evolve independently. OData defines semantics for both protocol and data model versioning.

5.1 Protocol Versioning

OData requests and responses are versioned according to the `OData-Version` header.

OData clients include the `OData-MaxVersion` header in requests in order to specify the maximum acceptable response version. Services respond with the maximum supported version that is less than or equal to the requested `OData-MaxVersion`, using decimal comparison. The syntax of the `OData-Version` and `OData-MaxVersion` header fields is specified in **[OData-ABNF]**.

This version of the specification defines data service version value `4.0`.

5.2 Model Versioning

The Data Model exposed by an OData Service defines a contract between the OData service and its clients. Services are allowed to extend their model only to the degree that it does not break existing clients. Breaking changes, such as removing properties or changing the type of existing properties, require that a new service version is provided at a different service root URL for the new model.

The following Data Model additions are considered safe and do not require services to version their entry point.

- Adding a property that is nullable or has a default value; if it has the same name as an existing dynamic property, it must have the same type (or base type) as the existing dynamic property
- Adding a navigation property that is nullable or collection-valued; if it has the same name as an existing dynamic navigation property, it must have the same type (or base type) as the existing dynamic navigation property
- Adding a new entity type to the model
- Adding a new complex type to the model
- Adding a new entity set
- Adding a new singleton
- Adding an action, a function, an action import, or function import
- Adding an action parameter that is nullable
- Adding a type definition or enumeration
- Adding any annotation to a model element that does not need to be understood by the client in order to correctly interact with the service

Clients SHOULD be prepared for services to make such incremental changes to their model. In particular, clients should be prepared to receive properties and derived types not previously defined by the service.

Services SHOULD NOT change their data model depending on the authenticated user. If the data model is user or user group dependent, all changes MUST be *safe changes* as defined in this section when comparing the full model to the model visible to users with restricted authorizations.

 Copyright © OASIS Open 2014. All Rights Reserved.

6 Extensibility

The OData protocol supports both user- and version-driven extensibility through a combination of versioning, convention, and explicit extension points.

6.1 Query Option Extensibility

Query options within the request URL can control how a particular request is processed by the service.

OData-defined system query options are prefixed with "`$`". Services may support additional custom query options not defined in the OData specification, but they MUST NOT begin with the "`$`" or "`@`" character.

OData services SHOULD NOT require any query options to be specified in a request. Services SHOULD fail any request that contains query options that they do not understand and MUST fail any request that contains unsupported OData query options defined in the version of this specification supported by the service.

In many cases OData services return URLs to identify resources that are later requested by clients. Where possible, interoperability is enhanced by providing all identifying information in the path portion of the URL. However, clients should be prepared for such URLs to include custom query options and propagate any such custom query options in future requests to the identified resource.

6.2 Payload Extensibility

OData supports extensibility in the payload, according to the specific format.

Regardless of the format, additional content MUST NOT be present if it needs to be understood by the receiver in order to correctly interpret the payload according to the specified `OData-Version` header. Thus, clients and services MUST be prepared to handle or safely ignore any content not specifically defined in the version of the payload specified by the `OData-Version` header.

6.3 Action/Function Extensibility

Actions and Functions extend the set of operations that can be performed on or with a service or resource. Actions can have side-effects. For example, Actions can be used to modify data or to invoke custom operations. Functions MUST NOT have side-effects. Functions can be invoked from a URL that addresses a resource or within an expression to a `$filter` or `$orderby` system query option.

Fully qualified action and function names include a namespace or alias prefix. The `Edm`, `odata` and `geo` namespaces are reserved for the use of this specification.

An OData service MUST fail any request that contains actions or functions that it does not understand.

6.4 Vocabulary Extensibility

The set of annotations defined within a schema comprise a *vocabulary*. Shared vocabularies provide a powerful extensibility point for OData.

Metadata annotations can be used to define additional characteristics or capabilities of a metadata element, such as a service, entity type, property, function, action or parameter. For example, a metadata annotation could define ranges of valid values for a particular property.

Instance annotations can be used to define additional information associated with a particular result, entity, property, or error; for example whether a property is read-only for a particular instance.

Where annotations apply across all instances of a type, services are encouraged to specify the annotation in metadata rather than repeating in each instance of the payload. Where the same annotation is defined at both the metadata and instance level, the instance-level annotation overrides the one specified at the metadata level.

 Copyright © OASIS Open 2014. All Rights Reserved.

A service MUST NOT require the client to understand custom annotations in order to accurately interpret a response.

OData defines a `Core` vocabulary with a set of basic terms describing behavioral aspects along with terms that can be used in defining other vocabularies; see **[OData-VocCore]**.

6.5 Header Field Extensibility

OData defines semantics around certain HTTP request and response headers. Services that support a version of OData conform to the processing requirements for the headers defined by this specification for that version.

Individual services may define custom headers. These headers MUST NOT begin with `OData` . Custom headers SHOULD be optional when making requests to the service. A service MUST NOT require the client to understand custom headers to accurately interpret the response.

6.6 Format Extensibility

An OData service MUST support at least one of **[OData-JSON]** or **[OData-Atom]**, and MAY support additional formats for both request and response bodies.

 Copyright © OASIS Open 2014. All Rights Reserved.

7 Formats

The client MAY request a particular response format through the `Accept` header, as specified in **[RFC7231]**, or through the system query option $format

In the case that both the `Accept` header and the `$format` query option are specified on a request, the value specified in the `$format` query option MUST be used.

If the service does not support the requested format, it replies with a `406 Not Acceptable` error response.

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

See the format specifications (**[OData-JSON]**, **[OData-Atom]**) for details.

Client libraries MUST retain the order of objects within an array in JSON responses, and elements in document order for ATOM and XML responses, including CSDL documents.

 Copyright © OASIS Open 2014. All Rights Reserved.

8 Header Fields

OData defines semantics around the following request and response headers. Additional headers may be specified, but have no unique semantics defined in OData.

8.1 Common Headers

The `Content-Type`, `Content-Length`, and `OData-Version` headers are common between OData requests and responses.

8.1.1 Header `Content-Type`

As specified in **[RFC7231],** the format of an individual request or response body MUST be specified in the `Content-Type` header of the request or response.

The specified format MAY include format parameters. Clients MUST be prepared for the service to return custom format parameters not specified in OData. Custom format parameters MUST NOT start with "odata" and services MUST NOT require generic OData consumers to understand custom format parameters in order to correctly interpret the payload.

See the format-specific specifications (**[OData-JSON]**, **[OData-Atom]**) for details.

8.1.2 Header `Content-Encoding`

As specified in **[RFC7231]**, the `Content-Encoding` header field is used as a modifier to the media-type (as indicated in the `Content-Type`). When present, its value indicates what additional content codings have been applied to the entity-body.
A service MAY specify a list of acceptable content codings using an annotation with term `Capabilities.AcceptableEncodings`, see **[OData-VocCap]**.

8.1.3 Header `Content-Language`

As specified in **[RFC7231],** a request or response can include a `Content-Language` header to indicate the natural language of the intended audience for the enclosed message body. OData does not add any additional requirements over HTTP for including `Content-Language`. OData services can annotate model elements whose content depends on the content language with the term `Core.IsLanguageDependent`, see **[OData-VocCore]**.

8.1.4 Header `Content-Length`

As specified in **[RFC7230],** a request or response SHOULD include a `Content-Length` header when the message's length can be determined prior to being transferred. OData does not add any additional requirements over HTTP for writing `Content-Length`.

8.1.5 Header `OData-Version`

OData clients SHOULD use the `OData-Version` header on a request to specify the version of the protocol used to generate the request.

If present on a request, the service MUST interpret the request according to the rules defined in the specified version of the protocol, or fail the request with a `4xx` response code.

If not specified in a request, the service MUST assume the request is generated using the minimum of the `OData-MaxVersion`, if specified, and the maximum version of the protocol that the service understands.

 Copyright © OASIS Open 2014. All Rights Reserved.

OData services MUST include the `OData-Version` header on a response to specify the version of the protocol used to generate the response. The client MUST interpret the response according to the rules defined in the specified version of the protocol.

For more details, see Versioning.

8.2 Request Headers

In addition to the Common Headers, the client may specify any combination of the following request headers.

8.2.1 Header `Accept`

As specified in **[RFC7231]**, the client MAY specify the set of accepted formats with the `Accept` Header.

Services MUST reject formats that specify unknown or unsupported format parameters.

If a media type specified in the `Accept` header includes a `charset` format parameter and the request also contains an `Accept-Charset` header, then the `Accept-Charset` header MUST be used.

If the media type specified in the `Accept` header does not include a `charset` format parameter, then the `Content-Type` header of the response MUST NOT contain a `charset` format parameter.

8.2.2 Header `Accept-Charset`

As specified in.**[RFC7231]**, the client MAY specify the set of accepted character sets with the `Accept-Charset` header.

8.2.3 Header `Accept-Language`

As specified in **[RFC7231]**, the client MAY specify the set of accepted natural languages with the `Accept-Language` header.

8.2.4 Header `If-Match`

As specified in **[RFC7232]**, a client MAY include an `If-Match` header in a request to `GET`, `PUT`, `PATCH` or `DELETE`. The value of the `If-Match` request header MUST be an ETag value previously retrieved for the entity, or `"*"` to match any value.

If an operation on an existing entity requires an ETag, (see `Core.OptimisticConcurrency` in **[OData-VocCore]**) and the client does not specify an `If-Match` request header in a Data Modification Request or in an Action Request bound to the entity, the service responds with a `428 Precondition Required` and MUST ensure that no observable change occurs as a result of the request.

If specified, the request MUST only be processed if the specified value matches the current ETag value of the target entity, using the weak comparison function (see **[RFC7232]**). If the value does not match the current ETag value of the entity for a Data Modification Request or Action Request, the service MUST respond with `412 Precondition Failed` and MUST ensure that no observable change occurs as a result of the request. In the case of an upsert, if the addressed entity does not exist the provided ETag value is considered not to match.

The client MAY include an `If-Match` header in a `PUT` or `PATCH` request in order to ensure that the request is handled as an update and not an upsert.

8.2.5 Header `If-None-Match`

As specified in **[RFC7232]**, a client MAY include an `If-None-Match` header in a request to `GET`, `PUT`, `PATCH` or `DELETE`. The value of the `If-None-Match` request header MUST be an ETag value previously retrieved for the entity, or `"*"`.

 Copyright © OASIS Open 2014. All Rights Reserved.

If specified, the request MUST only be processed if the specified value does not match the current ETag value of the entity, using the weak comparison function (see **[RFC7232]**). If the value matches the current ETag value of the entity, then for a `GET` request, the service SHOULD respond with `304 Not Modified`, and for a Data Modification Request or Action Request, the service MUST respond with `412 Precondition Failed` and MUST ensure that no observable change occurs as a result of the request.

An `If-None-Match` header with a value of `"*"` in a `PUT` or `PATCH` request results in an upsert request being processed as an insert and not an update.

8.2.6 Header `OData-Isolation`

The `OData-Isolation` header specifies the isolation of the current request from external changes. The only supported value for this header is `snapshot`.

If the service doesn't support `OData-Isolation:snapshot` and this header was specified on the request, the service MUST NOT process the request and MUST respond with `412 Precondition Failed.`

Snapshot isolation guarantees that all data returned for a request, including multiple requests within a batch or results retrieved across multiple pages, will be consistent as of a single point in time. Only data modifications made within the request (for example, by a data modification request within the same batch) are visible. The effect is as if the request generates a "snapshot" of the committed data as it existed at the start of the request.

The `OData-Isolation` header may be specified on a single or batch request. If it is specified on a batch then the value is applied to all statements within the batch.

Next links returned within a snapshot return results within the same snapshot as the initial request; the client is not required to repeat the header on each individual page request.

The `OData-Isolation` header has no effect on links other than the next link. Navigation links, read links, and edit links return the current version of the data.

A service returns `410 Gone` or `404 Not Found` if a consumer tries to follow a next link referring to a snapshot that is no longer available.

The syntax of the `OData-Isolation` header is specified in **[OData-ABNF]**.

A service MAY specify the support for `OData-Isolation:snapshot` using an annotation with term `Capabilities.IsolationSupport`, see **[OData-VocCap]**.

8.2.7 Header `OData-MaxVersion`

Clients SHOULD specify an `OData-MaxVersion` request header.

If specified the service MUST generate a response with an `OData-Version` less than or equal to the specified `OData-MaxVersion`.

If `OData-MaxVersion` is not specified, then the service SHOULD interpret the request as having an `OData-MaxVersion` equal to the maximum version supported by the service.

For more details, see Versioning.

8.2.8 Header `Prefer`

The `Prefer` header, as defined in **[RFC7240]**, allows clients to request certain behavior from the service. The service MUST ignore preference values that are either not supported or not know by the service.

The value of the `Prefer` header is a comma-separated list of *preferences*. The following subsections describe preferences whose meaning in OData is defined by this specification.

In response to a request containing a `Prefer` header, the service MAY return the `Preference-Applied` Header.

 Copyright © OASIS Open 2014. All Rights Reserved.

8.2.8.1 Preference odata.allow-entityreferences

The odata.allow-entityreferences preference indicates that the service is allowed to return entity references in place of entities that have previously been returned, with at least the properties requested, in the same response (for example, when serializing the expanded results of many-to-many relationships). The service MUST NOT return entity references in place of requested entities if odata.allow-entityreferences has not been specified in the request, unless explicitly defined by other rules in this document. The syntax of the odata.allow-entityreferences preference is specified in **[OData-ABNF]**.

In the case the service applies the odata.allow-entityreferences preference it MUST include a Preference-Applied response header containing the odata.allow-entityreferences preference to indicate that entity references MAY be returned in place of entities that have previously been returned.

8.2.8.2 Preference odata.callback

For scenarios in which links returned by the service are used by the client to poll for additional information, the client can specify the odata.callback preference to request that the service notify the client when data is available.

The odata.callback preference can be specified:

- when requesting asynchronous processing of a request with the respond-async preference, or
- on a GET request to a delta link.

The odata.callback preference MUST include the parameter url whose value is the URL of a callback endpoint to be invoked by the OData service when data is available. The syntax of the odata.callback preference is specified in **[OData-ABNF]**.

For HTTP based callbacks, the OData service executes an HTTP GET request against the specified URL.

Services that support odata.callback SHOULD support notifying the client through HTTP. Services can advertise callback support using the Capabilities.CallbackSupport annotation term defined in **[OData-VocCap]**.

If the service applies the odata.callback preference it MUST include the odata.callback preference in the Preference-Applied response header.

When the odata.callback preference is applied to asynchronous requests, the OData service invokes the callback endpoint once it has finished processing the request. The status monitor resource, returned in the Location header of the previously returned 202 Accepted response, can then be used to retrieve the results of the asynchronously executed request.

When the odata.callback preference is specified on a GET request to a delta link and there are no changes available, the OData service returns a 202 Accepted response with a Location header specifying the delta link to be used to check for future updates. The OData service then invokes the specified callback endpoint once new changes become available.

Combining respond-async, odata.callback and odata.track-changes preferences on a GET request to a delta-link might influence the response in a couple of ways.

- If the service processes the request synchronously, and no updates are available, then the response is the same as if the respond-async hadn't been specified and results in a response as described above.
- If the service processes the request asynchronously, then it responds with a 202 Accepted response specifying the URL to the status monitor resource as it would have with any other asynchronous request. Once the service has finished processing the asynchronous request to the delta link resource, if changes are available it invokes the specified callback endpoint. If no changes are available, the service SHOULD wait to notify the client until changes are available. Once notified, the client uses the status monitor resource from the Location header of the previously returned 202 Accepted response to retrieve the results. In case no updates were

 Copyright © OASIS Open 2014. All Rights Reserved.

available after processing the initial request, the result will contain no updates and the client can use the delta-link contained in the result to retrieve the updates that have since become available.

If the consumer specifies the same URL as callback endpoint in multiple requests, the service MAY collate them into a single notification once additional data is available for any of the requests. However, the consumer MUST be prepared to deal with receiving up to as many notifications as it requested.

Example 2: using a HTTP callback endpoint to receive notification

```
Prefer: odata.callback; url="http://myserver/notfication/token/12345"
```

8.2.8.3 Preference odata.continue-on-error

The odata.continue-on-error preference on a batch request is used to request that, upon encountering a request within the batch that returns an error, the service return the error for that request and continue processing additional requests within the batch. The syntax of the odata.continue-on-error preference is specified in **[OData-ABNF]**.

If not specified, upon encountering an error the service MUST return the error within the batch and stop processing additional requests within the batch.

A service MAY specify the support for the odata.continue-on-error preference using an annotation with term Capabilities.BatchContinueOnErrorSupported, see **[OData-VocCap]**.

8.2.8.4 Preference odata.include-annotations

The odata.include-annotations preference in a request for data or metadata is used to specify the set of annotations the client requests to be included, where applicable, in the response.

The value of the odata.include-annotations preference is a comma-separated list of namespaces or namespace qualified term names to include or exclude, with "*" representing all. The full syntax of the odata.include-annotations preference is defined in **[OData-ABNF]**.

The most specific identifier always takes precedence. If the same identifier value is requested to both be excluded and included the behavior is undefined; the service MAY return or omit the specified vocabulary but MUST NOT raise an exception.

Example 3: a `Prefer` header requesting all annotations within a metadata document to be returned

```
Prefer: odata.include-annotations="*"
```

Example 4: a `Prefer` header requesting that no annotations are returned

```
Prefer: odata.include-annotations="-*"
```

Example 5: a `Prefer` header requesting that all annotations defined under the "display" namespace (recursively) be returned

```
Prefer: odata.include-annotations="display.*"
```

Example 6: a `Prefer` header requesting that the annotation with the term name `subject` within the `display` namespace be returned if applied

```
Prefer: odata.include-annotations="display.subject"
```

The odata.include-annotations preference is only a hint to the service. The service MAY ignore the preference and is free to decide whether or not to return annotations not specified in the odata.include-annotations preference.

In the case that the client has specified the odata.include-annotations preference in the request, the service SHOULD include a Preference-Applied response header containing the odata.include-annotations preference to specify the annotations actually included, where applicable, in the response. This value may differ from the annotations requested in the Prefer header of the request.

 Copyright © OASIS Open 2014. All Rights Reserved.

8.2.8.5 Preference odata.maxpagesize

The odata.maxpagesize preference is used to request that each collection within the response contain no more than the number of items specified as the positive integer value of this preference. The syntax of the odata.maxpagesize preference is specified in **[OData-ABNF]**.

Example 7: a request for customers and their orders would result in a response containing one collection with customer entities and for every customer a separate collection with order entities. The client could specify odata.maxpagesize=50 in order to request that each page of results contain a maximum of 50 customers, each with a maximum of 50 orders.

If a collection within the result contains more than the specified odata.maxpagesize, the collection SHOULD be a partial set of the results with a next link to the next page of results. The client MAY specify a different value for this preference with every request following a next link.

In the example given above, the result page should include a next link for the customer collection, if there are more than 50 customers, and additional next links for all returned orders collections with more than 50 entities.

If the client has specified the odata.maxpagesize preference in the request, and the service limits the number of items in collections within the response through server-driven paging, the service MAY include a Preference-Applied response header containing the odata.maxpagesize preference and the maximum page size applied. This value may differ from the value requested by the client.

8.2.8.6 Preference odata.track-changes

The odata.track-changes preference is used to request that the service return a delta link that can subsequently be used to obtain changes (deltas) to this result. The syntax of the odata.track-changes preference is specified in **[OData-ABNF]**.

For paged results, the preference MUST be specified on the initial request. Services MUST ignore the odata.track-changes preference if applied to the next link.

The delta link MUST NOT be returned prior to the final page of results.

The service includes a Preference-Applied response header in the first page of the response containing the odata.track-changes preference to signal that changes are being tracked.

A service MAY specify the support for the odata.track-changes preference using an annotation with term Capabilities.ChangeTrackingSupport, see **[OData-VocCap]**.

8.2.8.7 Preference return=representation and return=minimal

The return=representation and return=minimal preferences are defined in **[RFC7240]**.

In OData, return=representation or return=minimal is defined for use with a POST, PUT, or PATCH Data Modification Request other than to a stream property, or to an Action Request. Specifying a preference of return=representation or return=minimal in a GET or DELETE request, or any request to a stream property, SHOULD return a 4xx Client Error.

A preference of return=representation or return=minimal is allowed on an individual Data Modification Request or Action Request within a batch, subject to the same restrictions, but SHOULD return a 4xx Client Error if specified on the batch request itself.

A preference of return=minimal requests that the service invoke the request but does not return content in the response. The service MAY apply this preference by returning 204 No Content in which case it MAY include a Preference-Applied response header containing the return=minimal preference.

A preference of return=representation requests that the service invokes the request and returns the modified resource. The service MAY apply this preference by returning the representation of the successfully modified resource in the body of the response, formatted according to the rules specified for the requested format. In this case the service MAY include a Preference-Applied response header containing the return=representation preference.

 Copyright © OASIS Open 2014. All Rights Reserved.

8.2.8.8 Preference respond-async

The respond-async preference, as defined in **[RFC7240]**, allows clients to request that the service process the request asynchronously.

If the client has specified respond-async in the request, the service MAY process the request asynchronously and return a 202 Accepted response.

The respond-async preference MAY be used for batch requests, but the service MUST ignore the respond-async preference for individual requests within a batch request.

In the case that the service applies the respond-async preference it MUST include a Preference-Applied response header containing the respond-async preference.

A service MAY specify the support for the respond-async preference using an annotation with term Capabilities.AsynchronousRequestsSupported, see **[OData-VocCap]**.

Example 8: a service receiving the following header might choose to respond

- *asynchronously if the synchronous processing of the request will take longer than 10 seconds*
- *synchronously after 5 seconds*
- *asynchronously (ignoring the wait preference)*
- *synchronously after 15 seconds (ignoring respond-async preference and the wait preference)*

```
Prefer: respond-async, wait=10
```

8.2.8.9 Preference wait

The wait preference, as defined in **[RFC7240]**, is used to establish an upper bound on the length of time, in seconds, the client is prepared to wait for the service to process the request synchronously once it has been received.

If the respond-async preference is also specified, the client requests that the service respond asynchronously after the specified length of time.

If the respond-async preference has not been specified, the service MAY interpret the wait as a request to timeout after the specified period of time.

8.3 Response Headers

In addition to the Common Headers, the following response headers have defined meaning in OData.

8.3.1 Header ETag

A request that returns an individual resource MAY include an ETag header in the response. Services MUST include this header in such a response if they require it to be specified when modifying the resource.

The value specified in the ETag header may be specified in the If-Match or If-None-Match header of a subsequent Data Modification Request or Action Request in order to apply optimistic concurrency in updating, deleting, or invoking the action bound to the entity.

As OData allows multiple formats for representing the same structured information, services SHOULD use weak ETags that only depend on the format-independent entity state is recommended. A strong ETag MUST change whenever the representation of an entity changes, so it has to depend on the Content-Type, the Content-Language, and potentially other characteristics of the response.

An ETag header MAY also be returned on a metadata document request or service document request to allow the client subsequently to make a conditional request for the metadata or service document. Clients can also compare the value of the ETag header returned from a metadata document request to the

 Copyright © OASIS Open 2014. All Rights Reserved.

metadata ETag returned in a response in order to verify the version of the metadata used to generate that response.

8.3.2 Header `Location`

The `Location` header MUST be returned in the response from a Create Entity or Create Media Entity request to specify the edit URL, or for read-only entities the read URL, of the created entity, and in responses returning `202 Accepted` to specify the URL that the client can use to request the status of an asynchronous request.

8.3.3 Header `OData-EntityId`

A response to a create operation that returns `204 No Content` MUST include an `OData-EntityId` response header. The value of the header is the entity-id of the entity that was acted on by the request. The syntax of the `OData-EntityId` preference is specified in **[OData-ABNF]**.

8.3.4 Header `Preference-Applied`

In a response to a request that specifies a `Prefer` header, a service MAY include a `Preference-Applied` header, as defined in **[RFC7240]**, specifying how individual preferences within the request were handled.

The value of the `Preference-Applied` header is a comma-separated list of preferences applied in the response. For more information on the individual preferences, see the `Prefer` header.

8.3.5 Header `Retry-After`

A service MAY include a `Retry-After` header in `202 Accepted` and in `3xx Redirect` responses

The `Retry-After` header specifies the duration of time, in seconds, that the client is asked to wait before retrying the request or issuing a request to the resource returned as the value of the `Location` header.

 Copyright © OASIS Open 2014. All Rights Reserved.

9 Common Response Status Codes

An OData service MAY respond to any request using any valid HTTP status code appropriate for the request. A service SHOULD be as specific as possible in its choice of HTTP status codes.

The following represent the most common success response codes. In some cases, a service MAY respond with a more specific success code.

9.1 Success Responses

The following response codes represent successful requests.

9.1.1 Response Code `200 OK`

A request that does not create a resource returns `200 OK` if it is completed successfully and the value of the resource is not `null`. In this case, the response body MUST contain the value of the resource specified in the request URL.

9.1.2 Response Code `201 Created`

A Create Entity, Create Media Entity, Create Link or Invoke Action request that successfully creates a resource returns `201 Created`. In this case, the response body MUST contain the resource created.

9.1.3 Response Code `202 Accepted`

`202 Accepted` indicates that the Data Service Request has been accepted and has not yet completed executing asynchronously. The asynchronous handling of requests is specified in section 11.6, and in section 11.7.5 for batch requests.

9.1.4 Response Code `204 No Content`

A request returns `204 No Content` if the requested resource has the `null` value, or if the service applies a `return=minimal` preference. In this case, the response body MUST be empty.

As defined in **[RFC7231]**, a Data Modification Request that responds with `204 No Content` MUST NOT include an `ETag` header unless the request's representation data was saved without any transformation applied to the body (i.e., the resource's new representation data is identical to the representation data received in the `PUT` request) and the `ETag` value reflects the new representation.

9.1.5 Response Code `3xx Redirection`

As per **[RFC7231]**, a `3xx Redirection` indicates that further action needs to be taken by the client in order to fulfill the request. In this case, the response SHOULD include a `Location` header, as appropriate, with the URL from which the result can be obtained; it MAY include a `Retry-After` header.

9.1.6 Response Code `304 Not Modified`

As per **[RFC7231]**, a `304 Not Modified` is returned when the client performs a `GET` request containing an `If-None-Match` header and the content has not changed. In this case the response SHOULD NOT include other headers in order to prevent inconsistencies between cached entity-bodies and updated headers.

The service MUST ensure that no observable change has occurred to the state of the service as a result of any request that returns a `304 Not Modified`.

 Copyright © OASIS Open 2014. All Rights Reserved.

9.2 Client Error Responses

Error codes in the `4xx` range indicate a client error, such as a malformed request.

The service MUST ensure that no observable change has occurred to the state of the service as a result of any request that returns an error status code.

In the case that a response body is defined for the error code, the body of the error is as defined for the appropriate format.

9.2.1 Response Code `404 Not Found`

`404 Not Found` indicates that the resource specified by the request URL does not exist. The response body MAY provide additional information.

9.2.2 Response Code `405 Method Not Allowed`

`405 Method Not Allowed` indicates that the resource specified by the request URL does not support the request method. In this case the response MUST include an `Allow` header containing a list of valid request methods for the requested resource as specified in **[RFC7231]**.

9.2.3 Response Code `410 Gone`

`410 Gone` indicates that the requested resource is no longer available. This can happen if a client has waited too long to follow a delta link or a status-monitor-resource link, or a next link on a collection that was requested with snapshot isolation.

9.2.4 Response Code `412 Precondition Failed`

As specified in **[RFC7232]**, `412 Precondition Failed` indicates that the client has performed a conditional request and the resource fails the condition. The service MUST ensure that no observable change occurs as a result of the request.

9.3 Server Error Responses

As specified in **[RFC7231]**, error codes in the `5xx` range indicate service errors.

9.3.1 Response Code `501 Not Implemented`

If the client requests functionality not implemented by the OData Service, the service MUST respond with `501 Not Implemented` and the response body SHOULD describe the functionality not implemented.

9.4 In-Stream Errors

In the case that the service encounters an error after sending a success status to the client, the service MUST generate an error within the payload, which may leave the response malformed. Clients MUST treat the entire response as being in error.

This specification does not prescribe a particular format for generating errors within a payload.

 Copyright © OASIS Open 2014. All Rights Reserved.

10 Context URL

The *context URL* describes the content of the payload. It consists of the canonical metadata document URL and a fragment identifying the relevant portion of the metadata document.

Request payloads generally do not require context URLs as the type of the payload can generally be determined from the request URL.

For details on how the context URL is used to describe a payload, see the relevant sections in the particular format.

The following subsections describe how the context URL is constructed for each category of payload by providing a *context URL template*. The context URL template uses the following terms:

- `{context-url}` is the canonical resource path to the `$metadata` document,
- `{entity-set}` is the name of an entity set or path to a containment navigation property,
- `{entity}` is the canonical URL for an entity,
- `{singleton}` is the canonical URL for a singleton entity,
- `{select-list}` is an optional parenthesized comma-separated list of selected properties, functions and actions,
- `{property-path}` is the path to a structural property of the entity,
- `{type-name}` is a qualified type name,
- `{/type-name}` is an optional type-cast segment containing the qualified name of a derived type prefixed with a forward slash.

The full grammar for the context URL is defined in **[OData-ABNF]**.

10.1 Service Document

Context URL template:

```
{context-url}
```

The context URL of the service document is the metadata document URL of the service.

Example 9: resource URL and corresponding context URL

```
http://host/service/
http://host/service/$metadata
```

10.2 Collection of Entities

Context URL template:

```
{context-url}#{entity-set}
```

```
{context-url}#Collection({type-name})
```

If all entities in the collection are members of one entity set, its name is the context URL fragment.

Example 10: resource URL and corresponding context URL

```
http://host/service/Customers
http://host/service/$metadata#Customers
```

If the entities are contained, then `entity-set` is the top-level entity set followed by the path to the containment navigation property of the containing entity.

Example 11: resource URL and corresponding context URL for contained entities

 Copyright © OASIS Open 2014. All Rights Reserved.

```
http://host/service/Orders(4711)/Items
http://host/service/$metadata#Orders(4711)/Items
```

If the entities in the response are not bound to a single entity set, such as from a function or action with no entity set path, a function import or action import with no specified entity set, or a navigation property with no navigation property binding, the context URL specifies the type of the returned entity collection.

10.3 Entity

Context URL template:

```
{context-url}#{entity-set}/$entity
```

```
{context-url}#{type-name}
```

If a response or response part is a single entity of the declared type of an entity set, `/$entity` is appended to the context URL.

Example 12: resource URL and corresponding context URL

```
http://host/service/Customers(1)
http://host/service/$metadata#Customers/$entity
```

If the entity is contained, then `entity-set` is the canonical URL for the containment navigation property of the containing entity, e.g. Orders(4711)/Items.

Example 13: resource URL and corresponding context URL for contained entity

```
http://host/service/Orders(4711)/Items(1)
http://host/service/$metadata#Orders(4711)/Items/$entity
```

If the response is not bound to a single entity set, such as an entity returned from a function or action with no entity set path, a function import or action import with no specified entity set, or a navigation property with no navigation property binding, the context URL specifies the type of the returned entity.

10.4 Singleton

Context URL template:

```
{context-url}#{singleton}
```

If a response or response part is a singleton, its name is the context URL fragment.

Example 14: resource URL and corresponding context URL

```
http://host/service/Contoso
http://host/service/$metadata#Contoso
```

10.5 Collection of Derived Entities

Context URL template:

```
{context-url}#{entity-set}{/type-name}
```

If an entity set consists exclusively of derived entities, a type-cast segment is added to the context URL.

Example 15: resource URL and corresponding context URL

```
http://host/service/Customers/Model.VipCustomer
http://host/service/$metadata#Customers/Model.VipCustomer
```

 Copyright © OASIS Open 2014. All Rights Reserved.

10.6 Derived Entity

Context URL template:

```
{context-url}#{entity-set}{/type-name}/$entity
```

If a response or response part is a single entity of a type derived from the declared type of an entity set, a type-cast segment is appended to the entity set name.

Example 16: resource URL and corresponding context URL

```
http://host/service/Customers(2)/Model.VipCustomer
http://host/service/$metadata#Customers/Model.VipCustomer/$entity
```

10.7 Collection of Projected Entities

Context URL templates:

```
{context-url}#{entity-set}{/type-name}{select-list}
```

```
{context-url}#Collection({type-name}){select-list}
```

If a result contains only a subset of properties, the parenthesized comma-separated list of the selected defined or dynamic properties, navigation properties, functions, and actions is appended to the `{entity-set}` after an optional type-cast segment, or the type of the entity collection if the response is not bound to a single entity set. The shortcut `*` represents the list of all structural properties. Properties defined on types derived from the declared type of the entity set (or type specified in the type-cast segment if specified) are prefixed with the qualified name of the derived type as defined in **[OData-ABNF]**.

Example 17: resource URL and corresponding context URL

```
http://host/service/Customers?$select=Address,Orders
http://host/service/$metadata#Customers(Address,Orders)
```

10.8 Projected Entity

Context URL templates:

```
{context-url}#{entity-set}{/type-name}{select-list}/$entity
```

```
{context-url}#{singleton}{select-list}
```

```
{context-url}#{type-name}{select-list}
```

If a single entity contains a subset of properties, the parenthesized comma-separated list of the selected defined or dynamic properties, navigation properties, functions, and actions is appended to the `{entity-set}` after an optional type-cast segment and prior to appending `/$entity`. If the response is not bound to a single entity set, the `{select-list}` is instead appended to the `{type-name}` of the returned entity.

The shortcut `*` represents the list of all structural properties. Properties defined on types derived from the type of the entity set (or type specified in the type-cast segment if specified) are prefixed with the qualified name of the derived type as defined in **[OData-ABNF]**. Note that expanded properties are implicitly selected.

Example 18: resource URL and corresponding context URL

```
http://host/service/Customers(1)?$select=Name,Rating
http://host/service/$metadata#Customers(Name,Rating)/$entity
```

 Copyright © OASIS Open 2014. All Rights Reserved.

10.9 Collection of Projected Expanded Entities

Context URL template:

```
{context-url}#{entity-set}{/type-name}{select-list}
```

```
{context-url}#Collection({type-name}){select-list}
```

If a navigation property is explicitly selected, the parenthesized comma-separated list of properties includes the name of the selected navigation property with no parenthesis. If a `$expand` contains a nested `$select`, the navigation property appears suffixed with the parenthesized comma-separated list of properties selected (or expanded, containing a `$select`) from the related entities. Additionally, if the expansion is recursive for nested children, a plus sign (+) is infixed between the navigation property name and the list of properties.

Example 19: resource URL and corresponding context URL

```
http://host/service/Customers$select=Name&$expand=Address/Country
http://host/service/$metadata#Customers(Name,Address/Country)
```

Example 20: resource URL and corresponding context URL

```
http://host/service/Employees/Sales.Manager?$select=DirectReports
                &$expand=DirectReports($select=FirstName,LastName;$levels=4)
http://host/service/$metadata
                #Employees/Sales.Manager(DirectReports,
                                         DirectReports+(FirstName,LastName))
```

10.10 Projected Expanded Entity

Context URL template:

```
{context-url}#{entity-set}{/type-name}{select-list}/$entity
```

```
{context-url}#{singleton}{select-list}
```

```
{context-url}#{type-name}{select-list}
```

If a single entity is expanded and projected (or contains a `$expand` with a `$select` expand option), the parenthesized comma-separated list of selected properties includes the name of the expanded navigation properties containing a nested `$select`, each suffixed with the parenthesized comma-separated list of properties selected (or expanded with a nested `$select`) from the related entities.

Example 21: resource URL and corresponding context URL

```
http://host/service/Employees(1)/Sales.Manager?
                  $expand=DirectReports($select=FirstName,LastName;$levels=4)
http://host/service/$metadata
       #Employees/Sales.Manager(DirectReports+(FirstName,LastName))/$entity
```

10.11 Collection of Entity References

Context URL template:

```
{context-url}#Collection($ref)
```

If a response is a collection of entity references, the context URL does not contain the type of the referenced entities.

Example 22: resource URL and corresponding context URL for a collection of entity references

```
http://host/service/Customers('ALFKI')/Orders/$ref
http://host/service/$metadata#Collection($ref)
```

 Copyright © OASIS Open 2014. All Rights Reserved.

10.12 Entity Reference

Context URL template:

```
{context-url}#$ref
```

If a response is a single entity reference, `$ref` is the context URL fragment.

Example 23: resource URL and corresponding context URL for a single entity reference

```
http://host/service/Orders(10643)/Customer/$ref
http://host/service/$metadata#$ref
```

10.13 Property Value

Context URL template:

```
{context-url}#{entity}/{property-path}{select-list}
```

If a response represents an individual property of an entity with a canonical URL, the context URL specifies the canonical URL of the entity and the path to the structural property of that entity. The path MUST include cast segments for properties defined on types derived from the expected type of the previous segment.

Example 24: resource URL and corresponding context URL

```
http://host/service/Customers(1)/Addresses
http://host/service/$metadata#Customers(1)/Addresses
```

10.14 Collection of Complex or Primitive Types

Context URL template:

```
{context-url}#Collection({type-name}){select-list}
```

If a response is a collection of complex types or primitive types that do not represent an individual property of an entity with a canonical URL, the context URL specifies the fully qualified type of the collection.

Example 25: resource URL and corresponding context URL

```
http://host/service/TopFiveHobbies()
http://host/service/$metadata#Collection(Edm.String)
```

10.15 Complex or Primitive Type

Context URL template:

```
{context-url}#{type-name}{select-list}
```

If a response is a complex type or primitive type that does not represent an individual property of an entity with a canonical URL, the context URL specifies the fully qualified type of the result.

Example 26: resource URL and corresponding context URL

```
http://host/service/MostPopularName()
http://host/service/$metadata#Edm.String
```

10.16 Operation Result

Context URL templates:

```
{context-url}#{entity-set}{/type-name}{select-list}
```

```
{context-url}#{entity-set}{/type-name}{select-list}/$entity
```

 Copyright © OASIS Open 2014. All Rights Reserved.

```
{context-url}#{entity}/{property-path}{select-list}
{context-url}#Collection({type-name}){select-list}
{context-url}#{type-name}{select-list}
```

If the response from an action or function is a collection of entities or a single entity that is a member of an entity set, the context URL identifies the entity set. If the response from an action or function is a property of a single entity, the context URL identifies the entity and property. Otherwise, the context URL identifies the type returned by the operation. The context URL will correspond to one of the former examples.

Example 27: resource URL and corresponding context URL

```
http://host/service/TopFiveCustomers{}
http://host/service/$metadata#Customers
```

10.17 Delta Response

Context URL template:

```
{context-url}#{entity-set}{/type-name}{select-list}/$delta
```

The context URL of a delta response is the same as the context URL of the root entity set, followed by `/$delta`.

Example 28: resource URL and corresponding context URL

```
http://host/service/Customers?$deltaToken=1234
http://host/service/$metadata#Customers/$delta
```

10.18 Item in a Delta Response

Context URL templates:

```
{context-url}#{entity-set}/$deletedEntity
{context-url}#{entity-set}/$link
{context-url}#{entity-set}/$deletedLink
```

In addition to new or changed entities which have the canonical context URL for an entity a delta response can contain deleted entities, new links, and deleted links. They are identified by the corresponding context URL fragment. `{entity-set}` corresponds to the set of the deleted entity, or source entity for an added or deleted link.

10.19 `$all` Response

Context URL template:

```
{context-url}#Collection(Edm.EntityType)
```

Responses to requests to the virtual collection `$all` (see **[OData-URL]**) use the built-in abstract entity type. Each single entity in such a response has its individual context URL that identifies the entity set or singleton.

10.20 `$crossjoin` Response

Context URL template:

```
{context-url}#Collection(Edm.ComplexType)
```

Responses to requests to the virtual collections `$crossjoin(...)` (see **[OData-URL]**) use the built-in abstract complex type. Single instances in these responses do not have a context URL.

 Copyright © OASIS Open 2014. All Rights Reserved.

11 Data Service Requests

11.1 Metadata Requests

An OData service is a self-describing service that exposes metadata defining the entity sets, relationships, entity types, and operations.

11.1.1 Service Document Request

Service documents enable simple hypermedia-driven clients to enumerate and explore the resources offered by the data service.

OData services MUST support returning a service document from the root URL of the service (the *service root*).

The format of the service document is dependent upon the format selected. For example, in Atom the service document is an AtomPub service document (as specified in **[**RFC5023],).

11.1.2 Metadata Document Request

An OData *Metadata Document* is a representation of the data model that describes the data and operations exposed by an OData service.

[OData-CSDL] describes an XML representation for OData metadata documents and provides an XML schema to validate their contents. The media type of the XML representation of an OData metadata document is `application/xml`.

OData services MUST expose a metadata document that describes the data model exposed by the service. The *Metadata Document URL* MUST be the root URL of the service with `$metadata` appended. To retrieve this document the client issues a `GET` request to the metadata document URL.

If a request for metadata does not specify a format preference (via `Accept` header or `$format`) then the XML representation MUST be returned.

11.1.3 Metadata Service Document Request

An OData Service MAY expose a Metadata Service. An OData *Metadata Service* is a representation of the data model that describes the data and operations exposed by an OData service as an OData service with a fixed (meta) data model.

A metadata service MUST use the schema defined in **[OData-CSDL]**. The root URL of the metadata service is the metadata document URL of the service with a forward slash appended. To retrieve this document the client issues a `GET` request to the metadata service root URL

11.2 Requesting Data

OData services support requests for data via HTTP `GET` requests.

The path of the URL specifies the target of the request (for example; the collection of entities, entity, navigation property, structural property, or operation). Additional query operators, such as filter, sort, page, and projection operations are specified through query options.

This section describes the types of data requests defined by OData. For complete details on the syntax for building requests, see **[OData-URL]**.

OData services are hypermedia driven services that return URLs to the client. If a client subsequently requests the advertised resource and the URL has expired, then the service SHOULD respond with `410 Gone`. If this is not feasible, the service MUST respond with `404 Not Found`.

 Copyright © OASIS Open 2014. All Rights Reserved.

The format of the returned data is dependent upon the request and the format specified by the client, either in the `Accept` header or using the `$format` query option.

11.2.1 Evaluating System Query Options

OData defines a number of system query options that allow refining the request. The result of the request MUST be as if the system query options were evaluated in the following order.

Prior to applying any server-driven paging:

- `$search`
- `$filter`
- `$count`
- `$orderby`
- `$skip`
- `$top`

After applying any server-driven paging:

- `$expand`
- `$select`
- `$format`

11.2.2 Requesting Individual Entities

To retrieve an individual entity, the client makes a `GET` request to the read URL of an entity.

The read URL can be obtained from a response payload containing that instance, for example as a self-link in an **[OData-Atom]** payload. In addition, Services MAY support conventions for constructing a read URL using the entity's key value(s), as described in **[OData-URL]**.

The set of structural or navigation properties to return may be specified through `$select` or `$expand` system query options.

Clients MUST be prepared to receive additional properties in an entity or complex type instance that are not advertised in metadata, even for types not marked as open.

Properties that are not available, for example due to permissions, are not returned. In this case, the `Core.Permissions` annotation, defined in **[OData-VocCore]** MUST be returned for the property with a value of `Core.Permission'None'`.

If no entity exists with the key values specified in the request URL, the service responds with `404 Not Found`.

11.2.3 Requesting Individual Properties

To retrieve an individual property, the client issues a `GET` request to the property URL. The property URL is the entity read URL with "/" and the property name appended.

For complex typed properties, the path can be further extended with the name of an individual property of the complex type.

See **[OData-URL]** for details.

If the property is single-valued and has the `null` value, the service responds with `204 No Content`.

If the property is not available, for example due to permissions, the service responds with `404 Not Found`.

Example 29:

```
http://host/service/Products(1)/Name
```

 Copyright © OASIS Open 2014. All Rights Reserved.

11.2.3.1 Requesting a Property's Raw Value using `$value`

To retrieve the raw value of a primitive type property, the client sends a `GET` request to the property value URL. See the **[OData-URL]** document for details.

The `Content-Type` of the response is determined using the `Accept` header and the `$format` system query option.

The default format for single primitive values except `Edm.Binary` and the `Edm.Geo` types is `text/plain`.

The default format for `Edm.Geo` types is `text/plain` using the WKT (well-known text) format, see rules `fullCollectionLiteral`, `fullLineStringLiteral`, `fullMultiPointLiteral`, `fullMultiLineStringLiteral`, `fullMultiPolygonLiteral`, `fullPointLiteral`, and `fullPolygonLiteral` in **[OData-ABNF]**.

The default format for `Edm.Binary` is the format specified by the `Core.MediaType` annotation of this property (see **[OData-VocCore]**) if this annotation is present. If not annotated, the format cannot be predicted by the client.

A `$value` request for a property that is `null` results in a `204 No Content` response.

If the property is not available, for example due to permissions, the service responds with `404 Not Found`.

Example 30:

```
http://host/service/Products(1)/Name/$value
```

11.2.4 Specifying Properties to Return

The `$select` and `$expand` system query options enable the client to specify the set of structural properties and navigation properties to include in a response. The service MAY include additional properties not specified in `$select` and `$expand`, including properties not defined in the metadata document.

11.2.4.1 System Query Option `$select`

The `$select` system query option requests that the service return only the properties, dynamic properties, actions and functions explicitly requested by the client. The service returns the specified content, if available, along with any available expanded navigation properties, and MAY return additional information.

The value of the `$select` query option is a comma-separated list of properties, qualified action names, qualified function names, the star operator (`*`), or the star operator prefixed with the namespace or alias of the schema in order to specify all operations defined in the schema.

Example 31: request only the `Rating` and `ReleaseDate` for the matching Products

```
http://host/service/Products?$select=Rating,ReleaseDate
```

It is also possible to request all structural properties, including any dynamic properties, using the star operator. The star operator SHOULD NOT introduce navigation properties, actions or functions not otherwise requested.

Example 32:

```
http://host/service/Products?$select=*
```

Properties of related entities can be specified by including the `$select` query option within the `$expand`.

Example 33:

```
http://host/service/Products?$expand=Category($select=Name)
```

 Copyright © OASIS Open 2014. All Rights Reserved.

The properties specified in `$select` are in addition to any expanded navigation properties.

Example 34: these two requests are equivalent

```
http://host/service/Categories?$select=CategoryName&$expand=Products
http://host/service/Categories?$select=CategoryName,Products&$expand=Products
```

It is also possible to request all actions or functions available for each returned entity.

Example 35:

```
http://host/service/Products?$select=DemoService.*
```

If the `$select` query option is not specified, the service returns the full set of properties or a default set of properties. The default set of properties MUST include all key properties.

If the service returns less than the full set of properties, either because the client specified a select or because the service returned a subset of properties in the absence of a select, the context URL MUST reflect the set of selected properties and expanded navigation properties.

11.2.4.2 System Query Option `$expand`

The `$expand` system query option indicates the related entities that MUST be represented inline. The service MUST return the specified content, and MAY choose to return additional information.

The value of the `$expand` query option is a comma-separated list of navigation property names, optionally followed by a `/$ref` path segment or a `/$count` path segment, and optionally a parenthesized set of expand options (for filtering, sorting, selecting, paging, or expanding the related entities).

For a full description of the syntax used when building requests, see **[OData-URL]**.

Example 36: for each customer entity within the Customers entity set the value of all related Orders will be represented inline

```
http://host/service.svc/Customers?$expand=Orders
```

Example 37: for each customer entity within the Customers entity set the references to the related Orders will be represented inline

```
http://host/service.svc/Customers?$expand=Orders/$ref
```

11.2.4.2.1 Expand Options

The set of expanded entities can be further refined through the application of expand options, expressed as a semicolon-separated list of system query options, enclosed in parentheses, see **[OData-URL]**.

Allowed system query options are `$filter`, `$select`, `$orderby`, `$skip`, `$top`, `$count`, `$search`, `$expand`, and `$levels`.

Example 38: for each customer entity within the `Customers` entity set, the value of those related `Orders` whose `Amount` is greater than 100 will be represented inline

```
http://host/service.svc/Customers?$expand=Orders($filter=Amount gt 100)
```

Example 39: for each order within the `Orders` entity set, the following will be represented inline:

- *The `Items` related to the `Orders` identified by the resource path section of the URL and the products related to each order item.*
- *The `Customer` related to each order returned.*

```
http://host/service.svc/Orders?$expand=Items($expand=Product),Customer
```

Example 40: for each customer entity in the Customers entity set, the value of all related InHouseStaff will be represented inline if the entity is of type VipCustomer or a subtype of that. For entities that are not of type

 Copyright © OASIS Open 2014. All Rights Reserved.

`VipCustomer`, or any of its subtypes, that entity may be returned with no inline representation for the expanded navigation property `InHouseStaff` (the service can always send more than requested)

```
http://host/service.svc/Customers?$expand=SampleModel.VipCustomer/InHouseStaff
```

11.2.4.2.1.1 Expand Option `$levels`

The `$levels` expand option can be used to specify the number of levels of recursion for a hierarchy in which the related entity type is the same as, or can be cast to, the source entity type. The same expand options are applied at each level of the hierarchy.

Services MAY support the symbolic value `max` in addition to numeric values. In that case they MUST solve circular dependencies by injecting an entity reference somewhere in the circular dependency.

Clients using `$levels=max` MUST be prepared to handle entity references in cases were a circular reference would occur otherwise.

Example 41: return each employee from the Employees entity set and, for each employee that is a manager, return all direct reports, recursively to four levels

```
http://contoso.com/HR/Employees?$expand=Model.Manager/DirectReports($levels=4)
```

11.2.5 Querying Collections

OData services support querying collections of entities, complex type instances, and primitive values.

The target collection is specified through a URL, and query operations such as filter, sort, paging, and projection are specified as *system query options* provided as query options. The names of all system query options are prefixed with a dollar (`$`) character.

The same system query option MUST NOT be specified more than once for any resource.

An OData service MAY support some or all of the system query options defined. If a data service does not support a system query option, it MUST fail any request that contains the unsupported option and SHOULD return `501 Not Implemented`.

11.2.5.1 System Query Option `$filter`

The `$filter` system query option restricts the set of items returned.

Example 42: return all Products whose `Price` is less than $10.00

```
http://host/service/Products?$filter=Price lt 10.00
```

The `$count` segment may be used within a `$filter` expression to limit the items returned based on the exact count of related entities or items within a collection-valued property.

Example 43: return all Categories with less than 10 products

```
http://host/service/Categories?$filter=Products/$count lt 10
```

The value of the `$filter` option is a Boolean expression as defined in **[OData-ABNF]**.

11.2.5.1.1 Built-in Filter Operations

OData supports a set of built-in filter operations, as described in this section. For a full description of the syntax used when building requests, see **[OData-URL]**.

 Copyright © OASIS Open 2014. All Rights Reserved.

Operator	Description	Example
Comparison Operators		
eq	Equal	Address/City eq 'Redmond'
ne	Not equal	Address/City ne 'London'
gt	Greater than	Price gt 20
ge	Greater than or equal	Price ge 10
lt	Less than	Price lt 20
le	Less than or equal	Price le 100
has	Has flags	Style has Sales.Color'Yellow'
Logical Operators		
and	Logical and	Price le 200 and Price gt 3.5
or	Logical or	Price le 3.5 or Price gt 200
not	Logical negation	not endswith(Description,'milk')
Arithmetic Operators		
add	Addition	Price add 5 gt 10
sub	Subtraction	Price sub 5 gt 10
mul	Multiplication	Price mul 2 gt 2000
div	Division	Price div 2 gt 4
mod	Modulo	Price mod 2 eq 0
Grouping Operators		
( )	Precedence grouping	(Price sub 5) gt 10

11.2.5.1.2 Built-in Query Functions

OData supports a set of built-in functions that can be used within `$filter` operations. The following table lists the available functions. For a full description of the syntax used when building requests, see **[OData-URL]**.

OData does not define an ISNULL or COALESCE operator. Instead, there is a `null` literal that can be used in comparisons.

Function	Example

 Copyright © OASIS Open 2014. All Rights Reserved.

Function	Example
String Functions	
contains	contains(CompanyName,'freds')
endswith	endswith(CompanyName,'Futterkiste')
startswith	startswith(CompanyName,'Alfr')
length	length(CompanyName) eq 19
indexof	indexof(CompanyName,'lfreds') eq 1
substring	substring(CompanyName,1) eq 'lfreds Futterkiste'
tolower	tolower(CompanyName) eq 'alfreds futterkiste'
toupper	toupper(CompanyName) eq 'ALFREDS FUTTERKISTE'
trim	trim(CompanyName) eq 'Alfreds Futterkiste'
concat	concat(concat(City,', '), Country) eq 'Berlin, Germany'
Date Functions	
year	year(BirthDate) eq 0
month	month(BirthDate) eq 12
day	day(StartTime) eq 8
hour	hour(StartTime) eq 1
minute	minute(StartTime) eq 0
second	second(StartTime) eq 0
fractionalseconds	second(StartTime) eq 0
date	date(StartTime) ne date(EndTime)
time	time(StartTime) le StartOfDay
totaloffsetminutes	totaloffsetminutes(StartTime) eq 60
now	StartTime ge now()
mindatetime	StartTime eq mindatetime()
maxdatetime	EndTime eq maxdatetime()
Math Functions	
round	round(Freight) eq 32
floor	floor(Freight) eq 32
ceiling	ceiling(Freight) eq 33

 Copyright © OASIS Open 2014. All Rights Reserved.

Function	Example
Type Functions	
cast	cast(ShipCountry,Edm.String)
isof	isof(NorthwindModel.Order)
isof	isof(ShipCountry,Edm.String)
Geo Functions	
geo.distance	geo.distance(CurrentPosition,TargetPosition)
geo.length	geo.length(DirectRoute)
geo.intersects	geo.intersects(Position,TargetArea)

11.2.5.1.3 Parameter Aliases

Parameter aliases can be used in place of literal values in function parameters or within a `$filter` or `$orderby` expression. Parameters aliases are names beginning with an at sign (`@`).

Actual parameter values are specified as query options in the query part of the request URL. The query option name is the name of the parameter alias, and the query option value is the value to be used for the specified parameter alias.

Example 44: returns all employees whose Region property matches the string parameter value "WA"

```
http://host/service.svc/Employees?$filter=Region eq @p1&@p1='WA'
```

Parameter aliases allow the same value to be used multiple times in a request and may be used to reference primitive values, complex, or collection values.

If a parameter alias is not given a value in the Query part of the request URL, the value MUST be assumed to be null. A parameter alias can be used in multiple places within a request URL but its value MUST NOT be specified more than once.

11.2.5.2 System Query Option `$orderby`

The `$orderby` System Query option specifies the order in which items are returned from the service.

The value of the `$orderby` System Query option contains a comma-separated list of expressions whose primitive result values are used to sort the items. A special case of such an expression is a property path terminating on a primitive property. A type cast using the qualified entity type name is required to order by a property defined on a derived type.

The expression can include the suffix `asc` for ascending or `desc` for descending, separated from the property name by one or more spaces. If `asc` or `desc` is not specified, the service MUST order by the specified property in ascending order.

Null values come before non-null values when sorting in ascending order and after non-null values when sorting in descending order.

Items are sorted by the result values of the first expression, and then items with the same value for the first expression are sorted by the result value of the second expression, and so on.

Example 45: return all Products ordered by release date in ascending order, then by rating in descending order

```
http://host/service/Products?$orderby=ReleaseDate asc, Rating desc
```

Related entities may be ordered by specifying `$orderby` within the `$expand` clause.

Example 46: return all Categories, and their Products ordered according to release date and in descending order of rating

 Copyright © OASIS Open 2014. All Rights Reserved.

```
http://host/service/Categories?
                        $expand=Products($orderby=ReleaseDate asc, Rating desc)
```

`$count` may be used within a `$orderby` expression to order the returned items according to the exact count of related entities or items within a collection-valued property.

Example 47: return all Categories ordered by the number of Products within each category

```
http://host/service/Cateoriges?$orderby=Products/$count
```

11.2.5.3 System Query Option `$top`

The `$top` system query option specifies a non-negative integer n that limits the number of items returned from a collection. The service returns the number of available items up to but not greater than the specified value n.

Example 48: return only the first five products of the Products entity set

```
http://host/service/Products?$top=5
```

If no unique ordering is imposed through an `$orderby` query option, the service MUST impose a stable ordering across requests that include `$top`.

11.2.5.4 System Query Option `$skip`

The `$skip` system query option specifies a non-negative integer n that excludes the first n items of the queried collection from the result. The service returns items starting at position n+1.

Example 49: return products starting with the 6th product of the `Products` entity set

```
http://host/service/Products?$skip=5
```

Where `$top` and `$skip` are used together, `$skip` MUST be applied before `$top`, regardless of the order in which they appear in the request.

Example 50: return the third through seventh products of the `Products` entity set

```
http://host/service/Products?$top=5&$skip=2
```

If no unique ordering is imposed through an `$orderby` query option, the service MUST impose a stable ordering across requests that include `$skip`.

11.2.5.5 System Query Option `$count`

The `$count` system query option with a value of `true` specifies that the total count of items within a collection matching the request be returned along with the result.

Example 51: return, along with the results, the total number of products in the collection

```
http://host/service/Products?$count=true
```

The count of related entities can be requested by specifying the `$count` query option within the `$expand` clause.

Example 52:

```
http://host/service/Categories?$expand=Products($count=true)
```

A `$count` query option with a value of `false` (or not specified) hints that the service SHOULD NOT return a count.

The service returns an HTTP Status code of `400 Bad Request` if a value other than `true` or `false` is specified.

 Copyright © OASIS Open 2014. All Rights Reserved.

The $count system query option ignores any $top, $skip, or $expand query options, and returns the total count of results across all pages including only those results matching any specified $filter and $search. Clients should be aware that the count returned inline may not exactly equal the actual number of items returned, due to latency between calculating the count and enumerating the last value or due to inexact calculations on the service.

How the count is encoded in the response body is dependent upon the selected format.

11.2.5.6 System Query Option $search

The $search system query option restricts the result to include only those entities *matching* the specified search expression. The definition of what it means to match is dependent upon the implementation.

Example 53: return all Products that match the search term "bike"

```
http://host/service/Products?$search=bike
```

The search expression can contain phrases, enclosed in double-quotes.

Example 54: return all Products that match the phrase "mountain bike"

```
http://host/service/Products?$search="mountain bike"
```

The upper case keyword NOT restricts the set of entities to those that do not match the specified term.

Example 55: return all Products that do not match "clothing"

```
http://host/service/Products?$search=NOT clothing
```

Multiple terms within a search expression are separated by a space (implicit AND) or the upper-case keyword AND, indicating that all such terms must be matched.

Example 56: return all Products that match both "mountain" and "bike"

```
http://host/service/Products?$search=mountain AND bike
```

The upper-case keyword OR is used to return entities that satisfy either the immediately preceding or subsequent expression.

Example 57: return all Products that match either "mountain" or "bike"

```
http://host/service/Products?$search=mountain OR bike
```

Parentheses within the search expression group together multiple expressions.

Example 58: return all Products that match either "mountain" or "bike" and do not match clothing

```
http://host/service/Products?$search=(mountain OR bike) AND NOT clothing
```

The operations within a search expression MUST be evaluated in the following order: grouping operator, NOT operator, AND operator, OR operator

If both $search and $filter are specified in the same request, only those entities satisfying both criteria are returned.

The value of the $search option is a Boolean expression as defined in **[OData-ABNF]**.

11.2.5.7 Server-Driven Paging

Responses that include only a partial set of the items identified by the request URL MUST contain a link that allows retrieving the next partial set of items. This link is called a *next link*; its representation is format-specific. The final partial set of items MUST NOT contain a next link.

The client can request a maximum page size through the odata.maxpagesize preference. The service may apply this requested page size or implement a page size different than, or in the absence of, this preference.

 Copyright © OASIS Open 2014. All Rights Reserved.

OData clients MUST treat the URL of the next link as opaque, and MUST NOT append system query options to the URL of a next link. Services may not allow a change of format on requests for subsequent pages using the next link. Clients therefore SHOULD request the same format on subsequent page requests using a compatible `Accept` header.OData services may use the reserved system query option `$skiptoken` when building next links. Its content is opaque, service-specific, and must only follow the rules for URL query parts.

OData clients MUST NOT use the system query option `$skiptoken` when constructing requests.

11.2.6 Requesting Related Entities

To request related entities according to a particular relationship, the client issues a `GET` request to the source entity's request URL, followed by a forward slash and the name of the navigation property representing the relationship.

If the navigation property does not exist on the entity indicated by the request URL, the service returns `404 Not Found`.

If the relationship terminates on a collection, the response MUST be the format-specific representation of the collection of related entities. If no entities are related, the response is the format-specific representation of an empty collection.

If the relationship terminates on a single entity, the response MUST be the format-specific representation of the related single entity. If no entity is related, the service returns `204 No Content`

Example 59: return the supplier of the product with `ID=1` in the Products entity set

```
http://host/service/Products(1)/Supplier
```

11.2.7 Requesting Entity References

To request entity references in place of the actual entities, the client issues a `GET` request with `/$ref` appended to the resource path.

If the resource path does not identify an entity or a collection of entities, the service returns `404 Not Found`.

If the resource path terminates on a collection, the response MUST be the format-specific representation of a collection of entity references pointing to the related entities. If no entities are related, the response is the format-specific representation of an empty collection.

If the resource path terminates on a single entity, the response MUST be the format-specific representation of an entity reference pointing to the related single entity. If the resource path terminates on a single entity and no such entity exists, the service returns `404 Not Found`.

Example 60: collection with an entity reference for each Order related to the Product with `ID=0`

```
http://host/service/Products(0)/Orders/$ref
```

11.2.8 Resolving an Entity-Id

To resolve an entity-id, e.g. obtained in an entity reference, into a representation of the identified entity, the client issues a `GET` request to the `$entity` resource which located at the URL `$entity` relative to the service root. The entity-id MUST be specified using the system query option `$id`.

Example 61: return the entity representation for a given entity-id

```
http://host/service/$entity?$id=http://host/service/Products(0)
```

A type segment following the `$entity` resource casts the resource to the specified type. If the identified entity is not of the specified type, or a type derived from the specified type, the service returns `404 Not Found`.

 Copyright © OASIS Open 2014. All Rights Reserved.

After applying a type-cast segment to cast to a specific type, the system query options `$select` and `$expand` can be specified in `GET` requests to the `$entity` resource.

Example 62: return the entity representation for a given entity-id and specify properties to return

```
http://host/service/$entity/Model.Customer?
                          $id=http://host/service/Customers('ALFKI')
                          &$select=CompanyName,ContactName&$expand=Orders
```

11.2.9 Requesting the Number of Items in a Collection

To request only the number of items of a collection of entities or items of a collection-valued property, the client issues a `GET` request with `/$count` appended to the resource path of the collection.

On success, the response body MUST contain the exact count of items matching the request after applying any `$filter` or `$search` system query options, formatted as a simple primitive integer value with media type `text/plain`. The returned count MUST NOT be affected by `$top`, `$skip`, `$orderby`, or `$expand`. Content negotiation using the `Accept` request header or the `$format` system query option is not allowed with the path segment `/$count`.

Example 63: return the number of products in the Products entity set

```
http://host/service/Products/$count
```

Example 64: return the number of all products whose `Price` is less than $10.00

```
http://host/service/Products/$count?$filter=Price lt 10.00
```

The `/$count` segment can be used in combination with the `$filter` system query option.

Example 65: return all customers with more than five interests

```
http://host/service/Customers?$filter=Interests/$count gt 5
```

11.2.10 System Query Option `$format`

The `$format` system query option specifies the media type of the response.

The `$format` query option, if present in a request, MUST take precedence over the value(s) specified in the `Accept` request header.

The value of the `$format` query option is a valid internet media type, optionally including parameters.

In addition, format-specific abbreviations may be used, see **[OData-Atom]** and **[OData-JSON]**, but format parameters MUST NOT be appended to the format abbreviations.

Example 66: the request

```
http://host/service/Orders?$format=application/json;odata.metadata=full
```

*is equivalent to a request with an `Accept` header using the same media type; it requests the set of Order entities represented using the JSON media type including full metadata, as specified in **[OData-JSON]**.*

Example 67: the request

```
http://host/service/Orders?$format=json
```

*is equivalent to a request with the `Accept` header set to `application/json`; it requests the set of Order entities represented using the JSON media type with minimal metadata, as specified in **[OData-JSON]**.*

The `$format` system query option MUST NOT be specified in batch requests as these always use the media type `multipart/mixed`.

Copyright © OASIS Open 2014. All Rights Reserved.

In metadata document requests the values `application/atom+xml`, `application/json`, their subtypes and parameterized variants as well as the format-specific abbreviations `atom` and `json` are reserved for future versions of this specification.

11.3 Requesting Changes

Services advertise their change-tracking capabilities by annotating entity sets with the `Capabilities.ChangeTracking` term defined in **[OData-VocCap]**.

Clients request that the service track changes to a result by specifying the `odata.track-changes` preference on a request. If supported for the request, the service includes a `Preference-Applied` header in the response containing the `odata.track-changes` preference and includes a *delta link* on the last page of results.

11.3.1 Delta Links

Delta links are opaque, service-generated links that the client uses to retrieve subsequent changes to a result.

Delta links are based on a *defining query* that describes the set of results for which changes are being tracked; for example, the request that generated the results containing the delta link. The delta link encodes the collection of entities for which changes are being tracked, along with a starting point from which to track changes.

If the defining query contains a `$filter` or `$search`, the response MUST include only changes to entities matching the specified criteria. Added entities MUST be returned for entities that were added or changed and now match the specified criteria, and deleted entities MUST be returned for entities that are changed to no longer match the criteria of `$filter` or `$search`.

The delta link MUST NOT encode any client top or skip value, and SHOULD NOT encode a request for an inline count.

If the defining query includes expanded relationships, the delta link MUST return changes, additions, or deletions to the expanded entities, as well as added or deleted links to expanded entities.

Entities are considered changed if any of the structural properties have changed. Changes to related entities and to streams are not considered a change to the entity containing the stream or navigation property.

If the defining query contains a projection, the generated delta link SHOULD logically include the same projection, such that the delta query only includes fields specified in the projection. Services MAY use the projection to limit the entities returned to those that have changed within the selected fields, but the client MUST be prepared to receive entities returned whether or not the field that changed was specified in the projection.

11.3.2 Using Delta Links

The client requests changes by invoking the `GET` method on the delta link. The client MUST NOT attempt to append system query options to the delta link. The `Accept` header MAY be used to specify the desired response format.

The `/$count` segment can be appended to the path of a delta link in order to get just the number of changes available. The count includes all added, changed, or deleted entities, as well as added or deleted links.

The results of a request against the delta link may span multiple pages but MUST be ordered by the service across all pages in such a way as to guarantee consistency when applied in order to the response which contained the delta link.

Services SHOULD return only changed entities, but MAY return additional entities matching the defining query for which the client will not see a change.

 Copyright © OASIS Open 2014. All Rights Reserved.

In order to continue tracking changes beyond the current set, the client specifies `odata.track-changes` on the initial request to the delta link but is not required to repeat it for subsequent pages. The new delta link appears at the end of the last page of changes and MUST return all changes subsequent to the last change of the previous delta link.

If no changes have occurred, the response is an empty collection that contains a delta link for subsequent changes if requested. This delta link MAY be identical to the delta link resulting in the empty collection of changes.

If the delta link is no longer valid, the service responds with `410 Gone`, and SHOULD include the URL for refetching the entire set in the `Location` header of the response.

11.4 Data Modification

Updatable OData services support Create, Update, and Delete operations for some or all exposed entities. Additionally, Actions supported by a service can affect the state of the system.

A successfully completed Data Modification Request must not violate the integrity of the data.

The client may request whether content be returned from a Create, Update, or Delete request, or the invocation of an Action, by specifying the `return` Prefer header.

11.4.1 Common Data Modification Semantics

Data Modification Requests share the following semantics.

11.4.1.1 Use of ETags for Avoiding Update Conflicts

If an ETag value is specified in an `If-Match` or `If-None-Match` header of a Data Modification Request or Action Request, the operation MUST only be invoked if the `if-match` or `if-none-match` condition is satisfied.

The ETag value specified in the `if-match` or `if-none-match` request header may be obtained from an ETag header of a response for an individual entity, or may be included for an individual entity in a format-specific manner.

11.4.1.2 Handling of `DateTimeOffset` Values

Services SHOULD preserve the offset of `Edm.DateTimeOffset` values, if possible. However, where the underlying storage does not support offset services may be forced to normalize the value to some common time zone (i.e. UTC) in which case the result would be returned with that time zone offset. If the service normalizes values, it MUST fail evaluation of the query functions `year`, `month`, `day`, `hour`, and `time` for literal values that are not stated in the time zone of the normalized values.

11.4.1.3 Handling of Properties Not Advertised in Metadata

Clients MUST be prepared to receive additional properties in an entity or complex type instance that are not advertised in metadata, even for types not marked as open. By using `PATCH` when updating entities, clients can ensure that such properties values are not lost if omitted from the update request.

11.4.1.4 Handling of Consistency Constraints

Services may impose cross-entity consistency constraints. Certain referential constraints, such as requiring an entity to be created with related entities can be satisfied through creating or linking related entities when creating the entity. Other constraints might require multiple changes to be specified together in a single atomic change set.

 Copyright © OASIS Open 2014. All Rights Reserved.

11.4.1.5 Returning Results from Data Modification Requests

Clients can request whether created or modified resources are returned from create, update, and upsert operations using the `return` preference header. In the absence of such a header, services SHOULD return the created or modified content unless the resource is a stream property value.

When returning content other than for an update to a media entity stream, services MUST return the same content as a subsequent request to retrieve the same resource. For updating media entity streams, the content of a non-empty response body MUST be the updated media entity.

11.4.2 Create an Entity

To create an entity in a collection, the client sends a `POST` request to that collection's URL. The `POST` body MUST contain a single valid entity representation.

An entity may also be created as the result of an Upsert operation.

If the target URL for the collection is a navigation link, the new entity is automatically linked to the entity containing the navigation link.

To create an *open entity* (an instance of an open type), additional property values beyond those specified in the metadata MAY be sent in the request body. The service MUST treat these as dynamic properties and add them to the created instance.

If the entity being created is not an open entity, additional property values beyond those specified in the metadata SHOULD NOT be sent in the request body. The service MUST fail if unable to persist all property values specified in the request.

Properties computed by the service (annotated with the term `Core.Computed`, see **[OData-VocCore]**) and properties that are tied to properties of the principal entity by a referential constraint, can be omitted and MUST be ignored if included in the request.

Upon successful completion, the response MUST contain a `Location` header that contains the edit URL or read URL of the created entity.

Upon successful completion the service MUST respond with either `201 Created`, or `204 No Content` if the request included a `return` Prefer header with a value of `return=minimal`.

11.4.2.1 Link to Related Entities When Creating an Entity

To create a new entity with links to existing entities in a single request, the client includes the entity-ids of the entities related through the corresponding navigation properties in the request body.

The representation for binding information is format-specific.

Example 68: using the JSON format the client can create a new manager entity with links to two existing employees by applying the `odata.bind` annotation to the `DirectReports` navigation property

```
{
  "@odata.type":"#Northwind.Manager",
  "EmployeeID": 1,
  "DirectReports@odata.bind": [
    "http://host/service/Employees(5)",
    "http://host/service/Employees(6)"
  ]
}
```

Example 69: using the Atom format the client can create a new manager entity with links to two existing employees by including a navigation link element for each employee in the Atom entry representing the manager

```
<entry>
  <id> http://host/service /Employees(1)</id>
  <title type="text" />
  <updated>2011-02-16T01:00:25Z</updated>
  <author><name /></author>
  <link rel="http://docs.oasis-open.org/odata/ns/related/DirectReports"
```

 Copyright © OASIS Open 2014. All Rights Reserved.

```
        href="http://host/service/Employees(5)"
        type="application/atom+xml;type=entry"
        title="Direct Reports" />
  <link rel="http://docs.oasis-open.org/odata/ns/related/DirectReports"
        href="http://host/service/Employees(6)"
        type="application/atom+xml;type=entry"
        title="Direct Reports" />
  <category term="NorthwindModel.Manager"
scheme="http://odata.org/scheme"/>
  <content type="application/xml">
    <metadata:properties>
      <data:EmployeeID>1</data:EmployeeID>
    </metadata:properties>
  </content>
</entry>
```

Upon successful completion of the operation, the service creates the requested entity and relates it to the requested existing entities.

If the target URL for the collection the entity is created in and binding information provided in the POST body contradicts the implicit binding information provided by the request URL, the request MUST fail and the service respond with `400 Bad Request`.

Upon failure of the operation, the service MUST NOT create the new entity. In particular, the service MUST never create an entity in a partially-valid state (with the navigation property unset).

11.4.2.2 Create Related Entities When Creating an Entity

A request to create an entity that includes related entities, represented using the appropriate inline representation, is referred to as a “deep insert”. Media entities, whose binary representation cannot be represented inline, cannot be created within a deep insert.

If the inline representation contains a value for a computed property or dependent property of a referential constraint, the service MUST ignore that value when creating the related entity.

On success, the service MUST create all entities and relate them. If the request included a `return` Prefer header with a value of `return=representation` and is applied by the service, the response MUST be expanded to at least the level that was present in the deep-insert request.

On failure, the service MUST NOT create any of the entities.

11.4.3 Update an Entity

Services SHOULD support `PATCH` as the preferred means of updating an entity. `PATCH` provides more resiliency between clients and services by directly modifying only those values specified by the client.

The semantics of `PATCH`, as defined in **[RFC5789]**, is to merge the content in the request payload with the [entity’s] current state, applying the update only to those components specified in the request body. Collection properties and primitive properties provided in the payload corresponding to updatable properties MUST replace the value of the corresponding property in the entity or complex type. Missing properties of the containing entity or complex property, including dynamic properties, MUST NOT be directly altered unless as a side effect of changes resulting from the provided properties.

Services MAY additionally support `PUT`, but should be aware of the potential for data-loss in round-tripping properties that the client may not know about in advance, such as open or added properties, or properties not specified in metadata. Services that support `PUT` MUST replace all values of structural properties with those specified in the request body. Missing non-key, updatable structural properties not defined as dependent properties within a referential constraint MUST be set to their default values. Omitting a non-nullable property with no service-generated or default value from a `PUT` request results in a `400 Bad Request` error. Missing dynamic structural properties MUST be removed or set to `null`.

 Copyright © OASIS Open 2014. All Rights Reserved.

Updating a dependent property that is tied to a key property of the principal entity through a referential constraint updates the relationship to point to the entity with the specified key value. If there is no such entity, the update fails.

Updating a principle property that is tied to a dependent entity through a referential constraint on the dependent entity updates the dependent property.

Key and other non-updatable properties, as well as dependent properties that are not tied to key properties of the principal entity, can be omitted from the request. If the request contains a value for one of these properties, the service MUST ignore that value when applying the update.

The service ignores entity id and entity type values in the payload when applying the update.

The entity MUST NOT contain related entities as inline content. It MAY contain binding information for navigation properties. For single-valued navigation properties this replaces the relationship. For collection-valued navigation properties this adds to the relationship.

If an update specifies both a binding to a single-valued navigation property and a dependent property that is tied to a key property of the principal entity according to the same navigation property, then the dependent property is ignored and the relationship is updated according to the value specified in the binding.

If the entity being updated is open, then additional values for properties beyond those specified in the metadata or returned in a previous request MAY be sent in the request body. The service MUST treat these as dynamic properties.

If the entity being updated is not open, then additional values for properties beyond those specified in the metadata or returned in a previous request SHOULD NOT be sent in the request body. The service MUST fail if it is unable to persist all updatable property values specified in the request.

On success, the response MUST be a valid success response.

11.4.4 Upsert an Entity

An upsert occurs when the client sends an update request to a valid URL that identifies a single entity that does not exist. In this case the service MUST handle the request as a create entity request or fail the request altogether.

Upserts are not supported against media entities or entities whose keys values are generated by the service. Services MUST fail an update request to a URL that would identify such an entity and the entity does not yet exist.

Key and other non-updatable properties, as well as dependent properties that are not tied to key properties of the principal entity, MUST be ignored by the service in processing the Upsert request.

To ensure that an update request is not treated as an insert, the client MAY specify an `If-Match` header in the update request. The service MUST NOT treat an update request containing an `If-Match` header as an insert.

A `PUT` or `PATCH` request MUST NOT be treated as an update if an `If-None-Match` header is specified with a value of `"*"`.

11.4.5 Delete an Entity

A successful `DELETE` request to an entity's edit URL deletes the entity. The request body SHOULD be empty. Singleton entities cannot be deleted.

On successful completion of the delete, the response MUST be `204 No Content` and contain an empty body.

Services MUST implicitly remove relations to and from an entity when deleting it; clients need not delete the relations explicitly.

Services MAY implicitly delete or modify related entities if required by integrity constraints. If integrity constraints are declared in `$metadata` using a `ReferentialConstraint` element, services MUST

 Copyright © OASIS Open 2014. All Rights Reserved.

modify affected related entities according to the declared integrity constraints, e.g. by deleting dependent entities, or setting dependent properties to `null` or their default value.

11.4.6 Modifying Relationships between Entities

Relationships between entities are represented by navigation properties as described in Data Model. URL conventions for navigation properties are described in **[OData-URL]**.

11.4.6.1 Add a Reference to a Collection-Valued Navigation Property

A successful `POST` request to a navigation property's references collection adds a relationship to an existing entity. The request body MUST contain a single entity reference that identifies the entity to be added. See the appropriate format document for details.

On successful completion, the response MUST be `204 No Content` and contain an empty body.

11.4.6.2 Remove a Reference to an Entity

A successful `DELETE` request to the URL that represents a reference to a related entity removes the relationship to that entity.

For collection-valued navigation properties, the entity reference of the entity to be removed MUST be specified using the `$id` query string option.

For single-valued navigation properties, the `$id` query string option MUST NOT be specified.

The `DELETE` request MUST NOT violate any integrity constraints in the data model.

On successful completion, the response MUST be `204 No Content` and contain an empty body.

11.4.6.3 Change the Reference in a Single-Valued Navigation Property

A successful `PUT` request to a single-valued navigation property's reference resource changes the related entity. The request body MUST contain a single entity reference that identifies the existing entity to be related. See the appropriate format document for details.

On successful completion, the response MUST be `204 No Content` and contain an empty body.

Alternatively, a relationship MAY be updated as part of an update to the source entity by including the required binding information for the new target entity. This binding information is format-specific, see **[OData-JSON]** and **[OData-Atom]** for details.

11.4.7 Managing Media Entities

A *media entity* is an entity that represents an out-of-band stream, such as a photograph.

A media entity MUST have a source URL that can be used to read the media stream, and MAY have a media edit URL that can be used to write to the media stream.

Because a media entity has both a media stream and standard entity properties special handling is required.

11.4.7.1 Creating a Media Entity

A `POST` request to a media entity's entity set creates a new media entity. The request body MUST contain the media value (for example, the photograph) whose media type MUST be specified in a `Content-Type` header.

Upon successful completion, the response MUST contain a `Location` header that contains the edit URL of the created entity.

Upon successful completion the service responds with either `201 Created`, or `204 No Content` if the request included a `return` Prefer header with a value of `return=minimal`.

 Copyright © OASIS Open 2014. All Rights Reserved.

11.4.7.2 Editing a Media Entity Stream

A successful `PUT` request to the media edit URL of a media entity changes the media stream of the entity.

If the entity includes an ETag value for the media stream, the client MUST include an `If-Match` header with the ETag value.

The request body MUST contain the new media value for the entity whose media type MUST be specified in a `Content-Type` header.

If the request to edit a media stream returns a non-empty response body, the response body MUST contain the updated media entity.

11.4.7.3 Deleting a Media Entity

A successful `DELETE` request to the entity's edit URL or to the edit URL of its media resource deletes the media entity as described in Delete An Entity.

Deleting a media entity also deletes the media associated with the entity.

11.4.8 Managing Stream Properties

An entity may have one or more *stream properties*. Stream properties are properties of type `Edm.Stream`.

The values for stream properties do not appear in the entity payload. Instead, the values are read or written through URLs.

11.4.8.1 Editing Stream Values

A successful `PUT` request to the edit URL of a stream property changes the media stream associated with that property.

If the stream metadata includes an ETag value, the client SHOULD include an `If-Match` header with the ETag value.

The request body MUST contain the new media value for the stream whose media type MUST be specified in a `Content-Type` header. It may have a `Content-Length` of zero to set the stream data to empty.

Stream properties MAY specify a list of acceptable media types using an annotation with term `Core.AcceptableMediaTypes`, see **[OData-VocCore]**.

11.4.8.2 Deleting Stream Values

A successful `DELETE` request to the edit URL of a stream property attempts to set the property to null and results in an error if the property is non-nullable.

Attempting to request a stream property whose value is null results in `204 No Content`.

11.4.9 Managing Values and Properties Directly

Values and properties can be explicitly addressed with URLs. The edit URL of a property is the edit URL of the entity appended with the path segment(s) specifying the individual property. The edit URL allows properties to be individually modified. See **[OData-URL]** for details on addressing.

11.4.9.1 Update a Primitive Property

A successful `PUT` request to the edit URL for a primitive property updates the value of the property. The message body MUST contain the new value, formatted as a single property according to the specified format.

 Copyright © OASIS Open 2014. All Rights Reserved.

A successful `PUT` request to the edit URL for the raw value of a primitive property updates the property with the raw value specified in the payload. The payload MUST be formatted as an appropriate content type for the raw value of the property.

The same rules apply whether this is a regular property or a dynamic property.

Upon successful completion of the update, the response MUST be a valid update response.

11.4.9.2 Set a Value to Null

A successful `DELETE` request to the edit URL for a structural property, or to the edit URL of the raw value of a primitive property, sets the property to null. The request body is ignored and should be empty.

A `DELETE` request to a non-nullable value MUST fail and the service respond with `400 Bad Request` or other appropriate error.

The same rules apply whether the target is the value of a regular property or the value of a dynamic property. A missing dynamic property is defined to be the same as a dynamic property with value `null`. All dynamic properties are nullable.

On success, the service MUST respond with `204 No Content` and an empty body.

Updating a primitive property or a complex property with a null value also sets the property to null.

11.4.9.3 Update a Complex Property

A successful `PATCH` request to the edit URL for a complex typed property updates that property. The request body MUST contain a single valid representation for the target complex type.

The service MUST directly modify only those properties of the complex type specified in the payload of the `PATCH` request.

The service MAY additionally support clients sending a `PUT` request to a URL that specifies a complex type. In this case, the service MUST replace the entire complex property with the values specified in the request body and set all unspecified properties to their default value.

On success, the response MUST be a valid update response.

11.4.9.4 Update a Collection Property

A successful `PUT` request to the edit URL of a collection property updates that collection. The message body MUST contain the desired new value, formatted as a collection property according to the specified format.

The service MUST replace the entire value with the value supplied in the request body.

Since collection members have no individual identity, `PATCH` is not supported for collection properties.

On success, the response MUST be a valid update response.

11.5 Operations

Custom operations (Actions and Functions) are represented as `Action`, `ActionImport`, `Function`, and `FunctionImport` elements in **[OData-CSDL]**.

11.5.1 Binding an Operation to a Resource

Actions and Functions MAY be bound to an entity type, primitive type, complex type, or a collection. The first parameter of a bound operation is the *binding parameter*.

The namespace- or alias-qualified name of a bound operation may be appended to any URL that identifies a resource whose type matches, or is derived from, the type of the binding parameter. The resource identified by that URL is used as the *binding parameter value*.

Example 70: the function `MostRecentOrder` can be bound to any URL that identifies a `SampleModel.Customer`

 Copyright © OASIS Open 2014. All Rights Reserved.

```
<Function Name="MostRecentOrder" ReturnType="SampleModel.Order"
                IsBound="true">
    <Parameter Name="customer" Type="SampleModel.Customer" />
</Function>
```

Example 71: invoking the `MostRecentOrder` function with the value of the binding parameter `customer` being the entity identified by `http://host/service/Customers(6)`

```
http://host/service/Customers(6)/SampleModel.MostRecentOrder()
```

11.5.2 Advertising Available Operations within a Payload

Services MAY return the available actions and/or functions bound to a particular entity as part of the entity representation within the payload. The representation of an action or function depends on the format. An efficient format that assumes client knowledge of metadata SHOULD NOT include actions and functions in the payload that are available on all instances and whose target URL can be computed via metadata following standard conventions defined in **[OData-URL]**.

Example 72: given a `GET` request to `http://host/service/Customers('ALFKI')`, the service might respond with a Customer that includes the `SampleEntities.MostRecentOrder` function bound to the entity

```
{
  "@odata.context": ...,
  "#SampleEntities.MostRecentOrder": {
    "title": "Most Recent Order",
    "target": "Customers('ALFKI')/SampleEntities.MostRecentOrder()"
  },
  "CustomerID": "ALFKI",
  "CompanyName": "Alfreds Futterkiste",
  ...
}
```

11.5.3 Functions

Functions are operations exposed by an OData service that MUST return data and MUST have no observable side effects.

11.5.3.1 Invoking a Function

To invoke a function bound to a resource, the client issues a `GET` request to a function URL. A function URL may be obtained from a previously returned entity representation or constructed by appending the namespace- or alias-qualified function name to a URL that identifies a resource whose type is the same as, or derived from, the type of the binding parameter of the function. The value for the binding parameter is the value of the resource identified by the URL prior to appending the function name, and additional parameter values are specified using inline parameter syntax. If the function URL is obtained from a previously returned entity representation, parameter aliases that are identical to the parameter name preceded by an at (`@`) sign MUST be used. 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 (`?`).

Functions can be used within `$filter` or `$orderby` system query options. Such functions can be bound to a resource, as described above, or called directly by specifying the namespace- (or alias-) qualified function name. Parameter values for functions within `$filter` or `$orderby` are specified according to the inline parameter syntax.

To invoke a function through a function import the client issues a `GET` request to a URL identifying the function import and passing parameter values using inline parameter syntax. The canonical URL for a function import is the service root, followed by the name of the function import.

 Copyright © OASIS Open 2014. All Rights Reserved.

Additional path segments and system query options may be appended to a URL or a path segment that identifies a composable function (or function import) as appropriate for the type returned by the function (or function import).

Services MAY support invoking a function or function import using a `PUT`, `POST`, `PATCH` or `DELETE` requests where the target of the operation is the result of the function.

Example 73: add a new item to the list of items of the shopping cart returned by the composable `MyShoppingCart` function import

```
POST http://host/service/MyShoppingCart()/Items

...
```

Parameter values passed to functions MUST be specified either as a URL literal (for primitive types) or as a JSON formatted OData object (for complex types or collections of primitive types or complex types).

If a collection-valued function has no result for a given parameter value combination, the response is the format-specific representation of an empty collection. If a single-valued function with a nullable return-type has no result, the service returns `204 No Content`.

If a single-valued function with a non-nullable return type has no result, the service returns `4xx`. For functions that return a single entity `404 Not Found` is the appropriate response code.

For a composable function the processing is stopped when the function result requires a `4xx` response, and continues otherwise.

Function imports MUST NOT be used inside either the `$filter` or `$orderby` system query options.

11.5.3.1.1 Inline Parameter Syntax

Parameter values are specified inline by appending a comma-separated list of parameter values, enclosed by parenthesis to the function name.

Each parameter value is represented as a name/value pair in the format `Name=Value`, where `Name` is the name of the parameter to the function and `Value` is the parameter value.

Example 74: invoke a `Sales.EmployeesByManager` function which takes a single `ManagerID` parameter via the function import `EmployeesByManager`

```
http://host/service/EmployeesByManager(ManagerID=3)
```

Example 75: return all `Customers` whose City property returns "Western" when passed to the `Sales.SalesRegion` function

```
http://host/service/Customers?
                          $filter=Sales.SalesRegion(City=$it/City) eq 'Western'
```

A parameter alias can be used in place of an inline parameter to a function call. The value for the alias is specified as a separate query option using the name of the parameter alias.

Example 76: invoke a `Sales.EmployeesByManager` function via the function import `EmployeesByManager`, passing 3 for the `ManagerID` parameter

```
http://host/service/EmployeesByManager(ManagerID=@p1)?@p1=3
```

11.5.3.2 Function overload resolution

The same function name may be used multiple times within a schema, each with a different set of parameters. For unbound overloads the combination of the function name and the unordered list of parameter types and names MUST identify a particular function overload. For bound overloads the combination of the function name, the binding parameter type, and the unordered set of names of the non-binding parameters MUST identify a particular function overload.

 Copyright © OASIS Open 2014. All Rights Reserved.

All unbound overloads MUST have the same return type. Also, all bound overloads with a given binding parameter type MUST have the same return type.

If the function is bound and the binding parameter type is part of an inheritance hierarchy, the function overload is selected based on the type of the URL segment preceding the function name. A type-cast segment can be used to select a function defined on a particular type in the hierarchy, see **[OData-URL]**.

11.5.4 Actions

Actions are operations exposed by an OData service that MAY have side effects when invoked. Actions MAY return data but MUST NOT be further composed with additional path segments.

11.5.4.1 Invoking an Action

To invoke an action bound to a resource, the client issues a `POST` request to an action URL. An action URL may be obtained from a previously returned entity representation or constructed by appending the namespace- or alias-qualified action name to a URL that identifies a resource whose type is the same as, or derives from, the type of the binding parameter of the action. The value for the binding parameter is the value of the resource identified by the URL prior to appending the action name, and any non-binding parameter values are passed in the request body according to the particular format.

To invoke an action through an action import, the client issues a `POST` request to a URL identifying the action import. The canonical URL for an action import is the service root, followed by the name of the action import. When invoking an action through an action import all parameter values MUST be passed in the request body according to the particular format.

Any nullable parameter values not specified in the request MUST be assumed to have the `null` value.

If the action returns results the client SHOULD use content type negotiation to request the results in the desired format, otherwise the default content type will be used.

To request processing of the action only if the binding parameter value, an entity or collection of entities, is unmodified, the client includes the `If-Match` header with the latest known ETag value for the entity or collection of entities. The ETag value for a collection as a whole is transported in the `ETag` header of a collection response.

On success, the response is `201 Created` for actions that create entities, `200 OK` for actions that return results or `204 No Content` for action without a return type. The client can request whether any results from the action be returned using the `Prefer` header.

Example 77: invoke the `SampleEntities.CreateOrder` action using `/Customers('ALFKI')` as the customer (or binding parameter). The values `2` for the `quantity` parameter and `BLACKFRIDAY` for the `discountCode` parameter are passed in the body of the request

```
POST http://host/service/Customers('ALFKI')/SampleEntities.CreateOrder

{
   "quantity": 2,
   "discountCode": "BLACKFRIDAY"
}
```

11.5.4.2 Action Overload Resolution

The same action name may be used multiple times within a schema provided there is at most one unbound overload, and each bound overload specifies a different binding parameter type.

If the action is bound and the binding parameter type is part of an inheritance hierarchy, the action overload is selected based on the type of the URL segment preceding the action name. A type-cast segment can be used to select an action defined on a particular type in the hierarchy, see **[OData-URL]**.

 Copyright © OASIS Open 2014. All Rights Reserved.

11.6 Asynchronous Requests

A Prefer header with a `respond-async` preference allows clients to request that the service process a Data Service Request asynchronously.

If the client has specified `respond-async` in the request, the service MAY process the request asynchronously and return a `202 Accepted` response. A service MUST NOT reply to a Data Service Request with `202 Accepted` if the request has not included the `respond-async` preference.

Responses that return `202 Accepted` MUST have an empty response body and MUST include a `Location` header pointing to a *status monitor resource* that represents the current state of the asynchronous processing in addition to an optional `Retry-After` header indicating the time, in seconds, the client should wait before querying the service for status.

A `GET` request to the status monitor resource again returns `202 Accepted` response if the asynchronous processing has not finished. This response MUST again include a `Location` header and MAY include a `Retry-After` header to be used for a subsequent request. The `Location` header and optional `Retry-After` header may or may not contain the same values as returned by the previous request.

A `GET` request to the status monitor resource returns `200 OK` once the asynchronous processing has completed. This response MUST include a `Content-Type` header with value `application/http` and a `Content-Transfer-Encoding` header with value `binary` as described in **[RFC7230]**. The response body MUST enclose a single HTTP response which is the response to the initial Data Service Request.

A `DELETE` request sent to the status monitor resource requests that the asynchronous processing be canceled. A `200 OK` or a `204 No Content` response indicates that the asynchronous processing has been successfully canceled. A client can request that the `DELETE` should be executed asynchronously. A `202 Accepted` response indicates that the cancellation is being processed asynchronously; the client can use the returned `Location` header (which MUST be different from the status monitor resource of the initial request) to query for the status of the cancellation. If a delete request is not supported by the service, the service returns `405 Method Not Allowed`.

After a successful `DELETE` request against the status monitor resource, any subsequent `GET` requests for the same status monitor resource returns `404 Not Found`.

If an asynchronous request is cancelled for reasons other than the consumers issuing a `DELETE` request against the status monitor resource, a `GET` request to the status monitor resource returns `200 OK` with a response body containing a single HTTP response with a status code in the `5xx Server Error` range indicating that the operation was cancelled.

The service MUST ensure that no observable change has occurred as a result of a canceled request.

If the client waits too long to request the result of the asynchronous processing, the service responds with a `410 Gone` or `404 Not Found`.

The status monitor resource URL MUST differ from any other resource URL.

11.7 Batch Requests

Batch requests allow grouping multiple operations into a single HTTP request payload. A batch request is represented as a Multipart MIME v1.0 message **[RFC2046]**, a standard format allowing the representation of multiple parts, each of which may have a different content type (as described in **[OData-Atom]** and **[OData-JSON]**), within a single request.

11.7.1 Batch Request Headers

Batch requests are submitted as a single HTTP `POST` request to the batch endpoint of a service, located at the URL `$batch` relative to the service root.

 Copyright © OASIS Open 2014. All Rights Reserved.

The batch request MUST contain a `Content-Type` header specifying a content type of `multipart/mixed` and a boundary specification as defined in **[RFC2046]** is defined in the Batch Request Body section below.

Batch requests SHOULD contain the applicable `OData-Version` header.

Example 78:

```
POST /service/$batch HTTP/1.1
Host: odata.org
OData-Version: 4.0
OData-MaxVersion: 4.0
Content-Type: multipart/mixed;boundary=batch_36522ad7-fc75-4b56-8c71-
56071383e77b

<Batch Request Body>
```

11.7.2 Batch Request Body

The body of a batch request is made up of a series of individual requests and change sets, each represented as a distinct MIME part (i.e. separated by the boundary defined in the `Content-Type` header).

The service MUST process the requests within a batch request sequentially. Processing stops on the first error unless the `odata.continue-on-error` preference is specified.

An individual request in the context of a batch request is a Data request, Data Modification request, Action invocation request, or Function invocation request. A MIME part representing an individual request MUST include a `Content-Type` header with value `application/http` and a `Content-Transfer-Encoding` header with value `binary`.

Preambles and Epilogues in the MIME payload, as defined in **[RFC2046]**, are valid but are assigned no meaning and thus MUST be ignored by processors of batch requests.

The Request-URI of HTTP requests serialized within MIME part bodies can use one of the following three formats:

- Absolute URI with schema, host, port, and absolute resource path.

  Example 79:

  ```
  GET https://host:1234/path/service/People(1) HTTP/1.1
  ```

- Absolute resource path and separate Host header.

  Example 80:

  ```
  GET /path/service/People(1) HTTP/1.1
  Host: myserver.mydomain.org:1234
  ```

- Resource path relative to the batch request URI.

  Example 81:

  ```
  GET People(1) HTTP/1.1
  ```

Services MUST support all three formats.

Each MIME part body that represents a single request MUST NOT include:

- `authentication` or `authorization` related HTTP headers
- `Expect`, `From`, `Max-Forwards`, `Range`, or `TE` headers

Processors of batch requests MAY choose to disallow additional HTTP constructs in HTTP requests serialized within MIME part bodies. For example, a processor may choose to disallow chunked encoding to be used by such HTTP requests.

Example 82: a batch request that contains the following operations in the order listed

 Copyright © OASIS Open 2014. All Rights Reserved.

1. *A query request*
2. *Change Set that contains the following requests:*
   - *Insert entity (with `Content-ID = 1`)*
   - *Update request (with `Content-ID = 2`)*
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.

Note also that the two empty lines after the `Host` header of the `GET` request are necessary: the first is part of the `GET` request header; the second is the empty body of the `GET` request, followed by a `CRLF` according to ***[RFC2046]****.*

```
POST /service/$batch HTTP/1.1
Host: host
OData-Version: 4.0
Content-Type: multipart/mixed;boundary=batch_36522ad7-fc75-4b56-8c71-
56071383e77b
Content-Length: ###

--batch_36522ad7-fc75-4b56-8c71-56071383e77b
Content-Type: application/http
Content-Transfer-Encoding:binary

GET /service/Customers('ALFKI')
Host: host


--batch_36522ad7-fc75-4b56-8c71-56071383e77b
Content-Type: multipart/mixed;boundary=changeset_77162fcd-b8da-41ac-a9f8-
9357efbbd

--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 1

POST /service/Customers HTTP/1.1
Host: host
Content-Type: application/atom+xml;type=entry
Content-Length: ###

<AtomPub representation of a new Customer>
--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
Content-Type: application/http
Content-Transfer-Encoding:binary
Content-ID: 2

PATCH /service/Customers('ALFKI') HTTP/1.1
Host: host
Content-Type: application/json
If-Match: xxxxx
Content-Length: ###

<JSON representation of Customer ALFKI>
```

 Copyright © OASIS Open 2014. All Rights Reserved.

```
--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd--
--batch_36522ad7-fc75-4b56-8c71-56071383e77b
Content-Type: application/http
Content-Transfer-Encoding: binary

GET /service/Products HTTP/1.1
Host: host


--batch_36522ad7-fc75-4b56-8c71-56071383e77b--
```

11.7.3 Change Sets

A *change set* is an atomic unit of work consisting of an unordered group of one or more Data Modification requests or Action invocation requests. Change sets MUST NOT contain any `GET` requests or other change sets. The contents of a MIME part representing a change set MUST itself be a multipart MIME document (see **[RFC2046]**) with one part for each operation that makes up the change set. Each part representing an operation in the change set MUST include the same headers (`Content-Type` and `Content-Transfer-Encoding`) and associated values as previously described for operations. In addition each request within a change set MUST specify a `Content-ID` header with a value unique within the batch request. The syntax of the `Content-ID` header is specified by rule `content-id` in **[OData-ABNF]**.

11.7.3.1 Referencing New Entities in a Change Set

Entities created by an Insert request within a change set can be referenced by subsequent requests within the same change set in places where a resource path to an existing entity can be specified. The temporary resource path for a newly inserted entity is the value of the `Content-ID` header prefixed with a `$` character. If `$<Content-ID>` is 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 the reference to the top-level system resource is used.

Example 83: a batch request that contains the following operations in the order listed:

A change set that contains the following requests:

 Copyright © OASIS Open 2014. All Rights Reserved.

- *Insert a new entity (with `Content-ID = 1`)*
- *Insert a second new entity (references request with `Content-ID = 1`)*

```
POST /service/$batch HTTP/1.1
Host: host
OData-Version: 4.0
Content-Type: multipart/mixed;boundary=batch_36522ad7-fc75-4b56-8c71-
56071383e77b

--batch_36522ad7-fc75-4b56-8c71-56071383e77b
Content-Type: multipart/mixed;boundary=changeset_77162fcd-b8da-41ac-a9f8-
9357efbbd

--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 1

POST /service/Customers HTTP/1.1
Host: host
Content-Type: application/atom+xml;type=entry
Content-Length: ###

<AtomPub representation of a new Customer>
--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 2

POST $1/Orders HTTP/1.1
Host: host
Content-Type: application/atom+xml;type=entry
Content-Length: ###

<AtomPub representation of a new Order>
--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd--
--batch_36522ad7-fc75-4b56-8c71-56071383e77b--
```

11.7.4 Responding to a Batch Request

Requests within a batch are evaluated according to the same semantics used when the request appears outside the context of a batch.

The order of change sets and individual requests in a Batch request is significant. A service MUST process the components of the Batch in the order received. The order of requests within a change set is not significant; a service may process the requests within a change set in any order.

All operations in a change set represent a single change unit so a service MUST successfully process and apply all the requests in the change set or else apply none of them. It is up to the service implementation to define rollback semantics to undo any requests within a change set that may have been applied before another request in that same change set failed and thereby apply this all-or-nothing requirement. The service MAY execute the requests within a change set in any order and MAY return the responses to the individual requests in any order. The service MUST include the `Content-ID` header in each response with the same value that the client specified in the corresponding request, so clients can correlate requests and responses.

If the set of request headers of a Batch request are valid (the `Content-Type` is set to `multipart/mixed`, etc.) the service MUST return a `200 OK` HTTP response code to indicate that the request was accepted for processing, but the processing is yet to be completed. The requests within the body of the batch may subsequently fail or be malformed; however, this enables batch implementations to stream the results.

 Copyright © OASIS Open 2014. All Rights Reserved.

If the service receives a Batch request with an invalid set of headers it MUST return a 4xx response code and perform no further processing of the request.

A response to a batch request MUST contain a `Content-Type` header with value `multipart/mixed`.

Structurally, a batch response body MUST match one-to-one with the corresponding batch request body, such that the same multipart MIME message structure defined for requests is used for responses. There are three exceptions to this rule:

- When a request within a change set fails, the change set response is not represented using the `multipart/mixed` media type. Instead, a single response, using the `application/http` media type and a `Content-Transfer-Encoding` header with a value of `binary`, is returned that applies to all requests in the change set and MUST be formatted according to the Error Handling defined for the particular response format.
- When an error occurs processing a request and the `odata.continue-on-error` preference is not specified, processing of the batch is terminated and the error response is the last part of the multi-part response.
- Asynchronously processed batch requests can return interim results and end with a `202 Accepted` as the last part of the multi-part response.

A response to an operation in a batch MUST be formatted exactly as it would have appeared outside of a batch as described in Requesting Data or Invoking a Function, as appropriate.

Example 84: referencing the batch request example 82 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: 4.0
Content-Length: ####
Content-Type: multipart/mixed;boundary=b_243234_25424_ef_892u748

--b_243234_25424_ef_892u748
Content-Type: application/http
Content-Transfer-Encoding: binary

HTTP/1.1 200 Ok
Content-Type: application/atom+xml;type=entry
Content-Length: ###

<AtomPub representation of the Customer entity with EntityKey ALFKI>
--b_243234_25424_ef_892u748
Content-Type: multipart/mixed;boundary=cs_12u7hdkin252452345eknd_383673037

--cs_12u7hdkin252452345eknd_383673037
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 1

HTTP/1.1 201 Created
Content-Type: application/atom+xml;type=entry
Location: http://host/service.svc/Customer('POIUY')
Content-Length: ###

<AtomPub representation of a new Customer entity>
```

 Copyright © OASIS Open 2014. All Rights Reserved.

```
--cs_12u7hdkin252452345eknd_383673037
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 2

HTTP/1.1 204 No Content
Host: host


--cs_12u7hdkin252452345eknd_383673037--
--b_243234_25424_ef_892u748
Content-Type: application/http
Content-Transfer-Encoding: binary

HTTP/1.1 404 Not Found
Content-Type: application/xml
Content-Length: ###

<Error message>
--b_243234_25424_ef_892u748--
```

11.7.5 Asynchronous Batch Requests

Batch requests may be executed asynchronously by including the `respond-async` preference in the Prefer header. The service MUST ignore the `respond-async` preference for individual requests within a batch.

After successful execution of the batch request the response to the batch request would be returned in the body of a response to an interrogation request against the *status monitor resource* URL (see section 11.6 “Asynchronous Requests”).

A service MAY return interim results to an asynchronously executing batch. It does this by including a `202 Accepted` response as the last part of the multi-part response. The client can use the monitor URL returned in this `202 Accepted` response to continue processing the batch response.

Since a change set is executed atomically, `202 Accepted` MUST NOT be returned within a change set.

Example 85: referencing the example 82 above again, assume that when interrogating the monitor URL for the first time only the first request in the batch finished processing and all the remaining requests except the final query request succeed. In this case the responses would be

```
HTTP/1.1 200 Ok
OData-Version: 4.0
Content-Length: ####
Content-Type: multipart/mixed;boundary=b_243234_25424_ef_892u748

--b_243234_25424_ef_892u748
Content-Type: application/http
Content-Transfer-Encoding: binary

HTTP/1.1 200 Ok
Content-Type: application/atom+xml;type=entry
Content-Length: ###

<AtomPub representation of the Customer entity with EntityKey ALFKI>
--b_243234_25424_ef_892u748
Content-Type: application/http
Content-Transfer-Encoding: binary

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

 Copyright © OASIS Open 2014. All Rights Reserved.

```
--b_243234_25424_ef_892u748--
```

Client makes a second request using the returned monitor URL

```
HTTP/1.1 200 Ok
OData-Version: 4.0
Content-Length: ####
Content-Type: multipart/mixed;boundary=b_243234_25424_ef_892u748

--b_243234_25424_ef_892u748
Content-Type: multipart/mixed;boundary=cs_12u7hdkin252452345eknd_383673037

--cs_12u7hdkin252452345eknd_383673037
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 1

HTTP/1.1 201 Created
Content-Type: application/atom+xml;type=entry
Location: http://host/service.svc/Customer('POIUY')
Content-Length: ###

<AtomPub representation of a new Customer entity>
--cs_12u7hdkin252452345eknd_383673037
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 2

HTTP/1.1 204 No Content
Host: host


--cs_12u7hdkin252452345eknd_383673037--
--b_243234_25424_ef_892u748
Content-Type: application/http
Content-Transfer-Encoding: binary

HTTP/1.1 404 Not Found
Content-Type: application/xml
Content-Length: ###

<Error message>
--b_243234_25424_ef_892u748—
```

 Copyright © OASIS Open 2014. All Rights Reserved.

12 Security Considerations

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 HTTP relevant security implications please cf. the relevant sections of **[RFC7231]** (9. Security Considerations) and for the HTTP `PATCH` method **[**RFC5789] (5. Security Considerations) as starting points.

12.1 Authentication

OData Services requiring authentication SHOULD consider supporting basic authentication as specified in **[**RFC2617] over HTTPS for the highest level of interoperability with generic clients. They MAY support other authentication methods.

 Copyright © OASIS Open 2014. All Rights Reserved.

13 Conformance

OData is designed as a set of conventions that can be layered on top of existing standards to provide common representations for common functionality. Not all services will support all of the conventions defined in the protocol; services choose those conventions defined in OData as the representation to expose that functionality appropriate for their scenarios.

To aid in client/server interoperability, this specification defines multiple levels of conformance for an OData Service, as well as the minimal requirements for an OData Client to be interoperable across OData services.

13.1 OData Service Conformance Levels

OData defines three levels of conformance for an OData Service.

Note: The conformance levels are design to correspond to different service scenarios. For example, a service that publishes data compliant with one or more of the OData defined formats may comply with the OData Minimal Conformance Level without supporting any additional functionality. A service that offers more control over the data that the client retrieves may comply with the OData Intermediate Conformance Level. Services that conform to the OData Advanced Conformance Level can expect to interoperate with the most functionality against the broadest range of generic clients. Services can advertise their level of conformance by the OData Conformance Level Annotation.

Services can advertise their level of conformance by annotating their entity container with the `Capabilities.ConformanceLevel` annotation defined in **[OData-VocCap]**.

Note: Services are encouraged to support as much additional functionality beyond their level of conformance as is appropriate for their intended scenario.

13.1.1 OData Minimal Conformance Level

In order to conform to the OData Minimal conformance level, a service:

1. MUST publish a service document at the service root (section 11.1.1)
2. MUST return data according to at least one of the OData defined formats (section 7)
3. MUST support server-driven paging when returning partial results (section 11.2.5.7)
4. MUST return the appropriate `OData-Version` header (section 8.1.5)
5. MUST conform to the semantics the following headers, or fail the request
   5.1. `Accept` (section 8.2.1)
   5.2. `OData-MaxVersion` (section 8.2.7)
6. MUST follow OData guidelines for extensibility (section 6 and all subsections)
7. MUST successfully parse the request according to **[OData-ABNF]** for any supported system query string options and either follow the specification or return `501 Not Implemented` (section 9.3.1) for any unsupported functionality (section 11.2.1)
8. MUST expose only data types defined in **[OData-CSDL]**
9. MUST NOT require clients to understand any metadata or instance annotations (section 6.4), custom headers (section 6.5), or custom content (section 6.2) in the payload in order to correctly consume the service
10. MUST NOT violate any OData update semantics (section 11.4 and all subsections)
11. MUST NOT violate any other OData-defined semantics
12. SHOULD support `$expand` (section 11.2.4.2)
13. MAY publish metadata at `$metadata` according to **[OData-CSDL]** (section 11.1.2)

In addition, to be considered an *Updatable OData Service*, the service:

14. MUST include edit links (explicitly or implicitly) for all updatable or deletable resources according to **[OData-Atom]** and **[OData-JSON]**

 Copyright © OASIS Open 2014. All Rights Reserved.

15. MUST support POST of new entities to insertable entity sets (section 11.4.1.5 and 11.4.2.1)
16. MUST support POST of new related entities to updatable navigation properties (section 11.4.6.1)
17. MUST support POST to `$ref` to add an existing entity to an updatable related collection (section 11.4.6.1)
18. MUST support PUT to `$ref` to set an existing single updatable related entity (section 11.4.6.3)
19. MUST support PATCH to all edit URLs for updatable resources (section 11.4.3)
20. MUST support DELETE to all edit URLs for deletable resources (section 11.4.5)
21. MUST support DELETE to `$ref` to remove an entity from an updatable navigation property (section 11.4.6.2)
22. MUST support if-match header in update/delete of any resources returned with an ETag (section 11.4.1.1)
23. MUST return a `Location` header with the edit URL or read URL of a created resource (section 11.4.1.5)
24. MUST include the `OData-EntityId` header in response to any POST/PATCH that returns `204 No Content` (Section 8.3.3)
25. MUST support Upserts (section 11.4.4)
26. SHOULD support PUT and PATCH to an individual primitive (section 11.4.9.1) or complex (section 11.4.9.3) property (respectively)
27. SHOULD support DELETE to set an individual property to null (section 11.4.9.2)
28. SHOULD support deep inserts (section 11.4.2.2)

13.1.2 OData Intermediate Conformance Level

In order to conform to the OData Intermediate Conformance Level, a service:

1. MUST conform to the OData Minimal Conformance Level
2. MUST successfully parse the **[OData-ABNF]** and either follow the specification or return `501 Not Implemented` for any unsupported functionality (section 9.3.1)
3. MUST support `$select` (section11.2.4.1)
4. MUST support casting to a derived type according to **[OData-URL]** if derived types are present in the model
5. MUST support `$top` (section 11.2.5.3)
6. MUST support `/$value` on media entities (section 4.10. in **[OData-URL]**) and individual properties (section 11.2.3.1)
7. MUST support `$filter` (section 11.2.5.1)
   7.1. MUST support `eq`, `ne` filter operations on properties of entities in the requested entity set (section 11.2.5.1.1)
   7.2. MUST support aliases in `$filter` expressions (section 11.2.5.1.3)
   7.3. SHOULD support additional filter operations (section 11.2.5.1.1) and MUST return `501 Not Implemented` for any unsupported filter operations (section 9.3.1)
   7.4. SHOULD support the canonical functions (section 11.2.5.1.2) and MUST return `501 Not Implemented` for any unsupported canonical functions (section 9.3.1)
   7.5. SHOULD support `$filter` on expanded entities (section 11.2.4.2.1)
8. SHOULD publish metadata at `$metadata` according to **[OData-CSDL]** (section 11.1.2)
9. SHOULD support the **[OData-JSON]** format
10. SHOULD consider supporting basic authentication as specified in **[RFC2617]** over HTTPS for the highest level of interoperability with generic clients
11. SHOULD support the `$search` system query option (section 11.2.5.6)
12. SHOULD support the `$skip` system query option (section 11.2.5.4)
13. SHOULD support the `$count` system query option (section 11.2.5.5)
14. SHOULD support `$expand` (section 11.2.4.2)
15. SHOULD support the lambda operators `any` and `all` on navigation- and collection-valued properties (section 5.1.1.5 in **[OData-URL]**)
16. SHOULD support the `/$count` segment on navigation and collection properties (section 11.2.9)

 Copyright © OASIS Open 2014. All Rights Reserved.

17. SHOULD support `$orderby asc` and `desc` on individual properties (section 11.2.5.2)

13.1.3 OData Advanced Conformance Level

In order to conform to the OData Advanced Conformance Level, a service:

1. MUST conform to at least the OData Intermediate Conformance Level
2. MUST publish metadata at `$metadata` according to [OData-CSDL] (section 11.1.2)
3. MUST support the **[OData-JSON]** format
4. MUST support the `/$count` segment on navigation and collection properties (section 11.2.9)
5. MUST support the lambda operators `any` and `all` on navigation- and collection-valued properties (section 5.1.1.5 in **[OData-URL]**)
6. MUST support the `$skip` system query option (section 11.2.5.4)
7. MUST support the `$count` system query option (section 11.2.5.5)
8. MUST support `$orderby asc` and `desc` on individual properties (section 11.2.5.2)
9. MUST support `$expand` (section 11.2.4.2)
   9.1. MUST support returning references for expanded properties (section 11.2.4.2)
   9.2. MUST support `$filter` on expanded entities (section 11.2.4.2.1)
   9.3. MUST support cast segment in expand with derived types (section 11.2.4.2.1)
   9.4. SHOULD support `$orderby asc` and `desc` on individual properties (section 11.2.4.2.1)
   9.5. SHOULD support the `$count` system query option for expanded properties (section 11.2.4.2.1)
   9.6. SHOULD support `$top` and `$skip` on expanded properties (section 11.2.4.2.1)
   9.7. SHOULD support `$search` on expanded properties (section 11.2.4.2.1)
   9.8. SHOULD support `$levels` for recursive expand (section 11.2.4.2.1.1)
10. MUST support the `$search` system query option (section 11.2.5.6)
11. MUST support batch requests (section11.7 and all subsections)
12. MUST support the resource path conventions defined in **[OData-URL]**
13. SHOULD support Asynchronous operations (section 8.2.8.8)
14. SHOULD support Delta change tracking (section 8.2.8.6)
15. SHOULD support cross-join queries defined in **[OData-URL]**
16. SHOULD support a conforming OData service interface over metadata (section 11.1.3)

13.2 Interoperable OData Clients

Interoperable OData Clients can expect to work with OData Services that comply with at least the OData Minimal Conformance Level and implement the **[OData-JSON]** format. Clients that additionally support **[OData-Atom]** can expect to interoperate with a broader range of OData Services.

To be generally interoperable, OData Clients

1. MUST specify the `OData-MaxVersion` header in requests (section 8.2.6)
2. MUST specify `OData-Version` (section 8.1.5) and `Content-Type` (section 8.1.1) in any request with a payload
3. MUST be a conforming consumer of OData as defined in **[OData-JSON]**
4. MUST follow redirects (section 9.1.5)
5. MUST correctly handle next links (section 11.2.5.7)
6. MUST support instances returning properties and navigation properties not specified in metadata (section 11.2)
7. MUST generate PATCH requests for updates, if the client supports updates (section 11.4.3)
8. SHOULD support basic authentication as specified in **[RFC2617]** over HTTPS
9. MAY request entity references in place of entities previously returned in the response (section 11.2.7)
10. MAY support deleted entities, link entities, deleted link entities in a delta response (section 11.3)
11. MAY support asynchronous responses (section 9.1.3)
12. MAY support `odata.metadata=minimal` in a JSON response (see **[OData-JSON]**)
13. MAY support `odata.streaming` in a JSON response (see **[OData-JSON]**)

 Copyright © OASIS Open 2014. All Rights Reserved.

Appendix A. Acknowledgments

The following individuals were members of the OASIS OData Technical Committee during the creation of this specification and their contributions are gratefully acknowledged:

- Howard Abrams (CA Technologies)
- Ken Baclawski (Northeastern University)
- Jay Balunas (Red Hat)
- Mark Biamonte (Progress Software)
- Matthew Borges (SAP AG)
- Edmond Bourne (BlackBerry)
- Joseph Boyle (Planetwork, Inc.)
- Peter Brown (Individual)
- Antonio Campanile (Bank of America)
- Pablo Castro (Microsoft)
- Axel Conrad (BlackBerry)
- Robin Cover (OASIS)
- Erik de Voogd (SDL)
- Diane Downie (Citrix Systems)
- Stefan Drees (Individual)
- Patrick Durusau (Individual)
- Andrew Eisenberg (IBM)
- Chet Ensign (OASIS)
- Davina Erasmus (SDL)
- Colleen Evans (Microsoft)
- Senaka Fernando (WSO2)
- Brent Gross (IBM)
- Zhun Guo (Individual)
- Anila Kumar GVN (CA Technologies)
- Ralf Handl (SAP AG)
- Barbara Hartel (SAP AG)
- Hubert Heijkers (IBM)
- Jens Hüsken (SAP AG)
- Evan Ireland (SAP AG)
- Gershon Janssen (Individual)
- Ram Jeyaraman (Microsoft)
- Ted Jones (Red Hat)
- Diane Jordan (IBM)
- Stephan Klevenz (SAP AG)
- Gerald Krause (SAP AG)
- Nuno Linhares (SDL)
- Paul Lipton (CA Technologies)
- Susan Malaika (IBM)
- Ramanjaneyulu Malisetti (CA Technologies)
- Neil McEvoy (iFOSSF – International Free and Open Source Solutions Foundation)
- Stan Mitranic (CA Technologies)
- Dale Moberg (Axway Software)
- Graham Moore (BrightstarDB Ltd.)
- Farrukh Najmi (Individual)
- Shishir Pardikar (Citrix Systems)
- Sanjay Patil (SAP AG)
- Nuccio Piscopo (iFOSSF – International Free and Open Source Solutions Foundation)
- Michael Pizzo (Microsoft)
- Robert Richards (Mashery)
- Sumedha Rubasinghe (WSO2)
- James Snell (IBM)
- Jeffrey Turpin (Axway Software)
- John Willson (Individual)
- John Wilmes (Individual)
- Christopher Woodruff (Perficient, Inc.)
- Martin Zurmuehl (SAP AG)

 Copyright © OASIS Open 2014. All Rights Reserved.

Appendix B. Revision History

Revision	Date	Editor	Changes Made
Working Draft 01	2012-08-22	Michael Pizzo	Translated Contribution to OASIS format/template
Committee Specification Draft 01	2013-04-26	Michael Pizzo, Ralf Handl, Martin Zurmuehl	Added Delta support, Asynchronous processing, Upsert Aligned and expanded Prefer header preferences Simplified data model Defined rules and semantics around distributed metadata Fleshed out descriptions and examples and addressed numerous editorial and technical issues processed through the TC Added Conformance section
Committee Specification Draft 02	2013-07-01	Michael Pizzo, Ralf Handl, Martin Zurmuehl	Cleaned up action and function overloads and binding, removed old-style function parameter syntax Improved asynchronous processing and added callback notifications Improved context URL (formerly: metadata URL) Improved handling of empty results Improved description of rules for processing PUT and POST requests, especially deep inserts Harmonized $count and $inlinecount
Committee Specification 01	2013-07-30	Michael Pizzo, Ralf Handl, Martin Zurmuehl	Non-Material Changes
Committee Specification Draft 03	2013-10-03	Michael Pizzo, Ralf Handl, Martin Zurmuehl	Improved description of type-cast rules

 Copyright © OASIS Open 2014. All Rights Reserved.