INTERNATIONAL STANDARD

ISO/IEC 20802-1

First edition
2016-12-15

Information technology — Open data protocol (OData) v4.0

Part 1: **Core**

Technologies de l'information — Protocole de données ouvertes (OData) v4.0 —

Partie 2: Base

Reference number
ISO/IEC 20802-1:2016(E)

COPYRIGHT PROTECTED DOCUMENT

ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT) see the following URL: www.iso.org/iso/foreword.html.

ISO/IEC 20802-1 was prepared by OASIS and was adopted, under the PAS procedure, by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, in parallel with its approval by the national bodies of ISO and IEC.

OData Version 4.0 Part 1: Protocol Plus Errata 02

OASIS Standard incorporating Approved Errata 02

30 October 2014

Specification URIs

This version:

http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part1-protocol/odata-v4.0-errata02-os-part1-protocol-complete.doc (Authoritative)
http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part1-protocol/odata-v4.0-errata02-os-part1-protocol-complete.html
http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part1-protocol/odata-v4.0-errata02-os-part1-protocol-complete.pdf

Previous 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

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 02*. Edited by Michael Pizzo, Ralf Handl, Martin Zurmuehl, and Hubert Heijkers. 30 October 2014. OASIS Approved Errata. http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/odata-v4.0-errata02-os.html.
- *OData Version 4.0 Part 1: Protocol Plus Errata 02* (this document). Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. 30 October 2014. OASIS Standard incorporating Approved Errata 02. http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part1-protocol/odata-v4.0-errata02-os-part1-protocol-complete.html.
- *OData Version 4.0 Part 2: URL Conventions Plus Errata 02*. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. 30 October 2014. OASIS Standard incorporating Approved

Errata 02. http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part2-url-conventions/odata-v4.0-errata02-os-part2-url-conventions-complete.html.

- *OData Version 4.0 Part 3: Common Schema Definition Language (CSDL) Plus Errata 02*. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. 30 October 2014. OASIS Standard incorporating Approved Errata 02. http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part3-csdl/odata-v4.0-errata02-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/errata02/os/complete/abnf/.
- Vocabulary components: OData Core Vocabulary, OData Measures Vocabulary and OData Capabilities Vocabulary. http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/vocabularies/.
- XML schemas: OData EDMX XML Schema and OData EDM XML Schema. http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/schemas/.
- OData Metadata Service Entity Model: http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/models/.
- Change-marked (redlined) versions of OData Version 4.0 Part 1, Part 2, and Part 3. OASIS Standard incorporating Approved Errata 02. http://docs.oasis-open.org/odata/odata/v4.0/errata02/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]

OData Version 4.0 Part 1: Protocol Plus Errata 02. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. 30 October 2014. OASIS Standard incorporating Approved Errata 02. http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part1-protocol/odata-v4.0-errata02-os-part1-protocol-complete.html. Latest version: http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part1-protocol.html.

Notices

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

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

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

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

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

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

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
11.2.5 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
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
Appendix A. Acknowledgments 72

Appendix B. Revision History 73

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.

[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.

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.

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.

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*.

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.

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.

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.

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]**ODataJSONRef**OData-JSON** or **[OData-Atom]OData-Atom**, and MAY support additional formats for both request and response bodies.

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-JSON**, **[OData-Atom]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.

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.

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 **[RFC7230]**). 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 `"*"`.

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 **[RFC7230]**). 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.

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

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.

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.

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

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.

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`.

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 **[RFC7230]**, `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.

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


```
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
```


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]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]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
```


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)
```


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
```



```
{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.

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`.

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
```

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)
```


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

`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]**.

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

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

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


```
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.

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.

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`.

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`.

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.

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.

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]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"
```



```
        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`.

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

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`.

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.

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`


```
<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.

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.

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]**.

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.

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

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>
```



```
--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:

- *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.

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>
```

```
--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: ###
```



```
--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—
```


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 **[RFC5023]** (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.

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]**

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 `/$count` segment on navigation and collection properties (section 11.2.9)
17. SHOULD support `$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]**)

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)

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
Committee Specification 02	2013-11-04	Michael Pizzo, Ralf Handl, Martin Zurmuehl	Non-Material Changes
OASIS Specification	2014-02-24	Michael Pizzo, Ralf Handl, Martin Zurmuehl	Non-Material Changes
Errata 01	2014-07-24	Michael Pizzo, Ralf Handl, Martin Zurmuehl	Minor changes and improvements

Errata 02	2014-10-29	Michael Pizzo, Ralf Handl, Martin Zurmuehl	Repaired mechanical error in the editable source

OData Version 4.0 Part 2: URL Conventions Plus Errata 02

OASIS Standard incorporating Approved Errata 02

30 October 2014

Specification URIs

This version:

http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part2-url-conventions/odata-v4.0-errata02-os-part2-url-conventions-complete.doc (Authoritative)
http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part2-url-conventions/odata-v4.0-errata02-os-part2-url-conventions-complete.html
http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part2-url-conventions/odata-v4.0-errata02-os-part2-url-conventions-complete.pdf

Previous version:

http://docs.oasis-open.org/odata/odata/v4.0/errata01/os/complete/part2-url-conventions/odata-v4.0-errata01-os-part2-url-conventions-complete.doc (Authoritative)
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
http://docs.oasis-open.org/odata/odata/v4.0/errata01/os/complete/part2-url-conventions/odata-v4.0-errata01-os-part2-url-conventions-complete.pdf

Latest version:

http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part2-url-conventions.doc (Authoritative)
http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part2-url-conventions.html
http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part2-url-conventions.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 02*. Edited by Michael Pizzo, Ralf Handl, Martin Zurmuehl, and Hubert Heijkers. 30 October 2014. OASIS Approved Errata. http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/odata-v4.0-errata02-os.html.
- *OData Version 4.0 Part 1: Protocol Plus Errata 02*. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. 30 October 2014. OASIS Standard incorporating Approved Errata 02. http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part1-protocol/odata-v4.0-errata02-os-part1-protocol-complete.html.
- *OData Version 4.0 Part 2: URL Conventions Plus Errata 02* (this document). Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. 30 October 2014. OASIS Standard

incorporating Approved Errata 02. http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part2-url-conventions/odata-v4.0-errata02-os-part2-url-conventions-complete.html.

- *OData Version 4.0 Part 3: Common Schema Definition Language (CSDL) Plus Errata 02*. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. 30 October 2014. OASIS Standard incorporating Approved Errata 02. http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part3-csdl/odata-v4.0-errata02-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/errata02/os/complete/abnf/.
- Vocabulary components: OData Core Vocabulary, OData Measures Vocabulary and OData Capabilities Vocabulary. http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/vocabularies/.
- XML schemas: OData EDMX XML Schema and OData EDM XML Schema. http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/schemas/.
- OData Metadata Service Entity Model: http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/models/.
- Change-marked (redlined) versions of OData Version 4.0 Part 1, Part 2, and Part 3. OASIS Standard incorporating Approved Errata 02. http://docs.oasis-open.org/odata/odata/v4.0/errata02/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 a data model, to be published and edited by Web clients using simple HTTP messages. This specification defines a set of recommended (but not required) rules for constructing URLs to identify the data and metadata exposed by an OData service as well as a set of reserved URL query string operators.

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-Part2]

OData Version 4.0 Part 2: URL Conventions Plus Errata 02. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. 30 October 2014. OASIS Standard incorporating Approved Errata 02. http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part2-url-conventions/odata-v4.0-errata02-os-part2-url-conventions-complete.html. Latest version: http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part2-url-conventions.html.

Notices

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

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

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

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

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 8
1.1 Terminology 8
1.2 Normative References 8
1.3 Typographical Conventions 8
2 URL Components 10
3 Service Root URL 12
4 Resource Path 13
4.1 Addressing the Model for a Service 13
4.2 Addressing the Batch Endpoint for a Service 13
4.3 Addressing Entities 13
4.3.1 Canonical URL 15
4.3.2 Canonical URL for Contained Entities 16
4.3.3 URLs for Related Entities with Referential Constraints 16
4.3.4 Resolving an Entity-Id 16
4.4 Addressing References between Entities 16
4.5 Addressing Operations 17
4.5.1 Addressing Actions 17
4.5.2 Addressing Functions 17
4.6 Addressing a Property 18
4.7 Addressing a Property Value 18
4.8 Addressing the Count of a Collection 18
4.9 Addressing Derived Types 18
4.10 Addressing the Media Stream of a Media Entity 19
4.11 Addressing the Cross Join of Entity Sets 19
4.12 Addressing All Entities in a Service 20
5 Query Options 21
5.1 System Query Options 21
5.1.1 System Query Option `$filter` 21
5.1.1.1 Logical Operators 21
5.1.1.1.1 Equals 22
5.1.1.1.2 Not Equals 22
5.1.1.1.3 Greater Than 22
5.1.1.1.4 Greater Than or Equal 22
5.1.1.1.5 Less Than 22
5.1.1.1.6 Less Than or Equal 22
5.1.1.1.7 And 22
5.1.1.1.8 Or 22
5.1.1.1.9 Not 22
5.1.1.1.10 `has` 23
5.1.1.1.11 Logical Operator Examples 23
5.1.1.2 Arithmetic Operators 23
5.1.1.2.1 Addition 23
5.1.1.2.2 Subtraction 24
5.1.1.2.3 Negation 24

5.1.1.2.4 Multiplication 24
5.1.1.2.5 Division....24
5.1.1.2.6 Modulo 24
5.1.1.2.7 Arithmetic Operator Examples 24
5.1.1.3 Grouping 25
5.1.1.4 Canonical Functions 25
5.1.1.4.1 `contains`....25
5.1.1.4.2 `endswith`....25
5.1.1.4.3 `startswith` 25
5.1.1.4.4 `length` 26
5.1.1.4.5 `indexof` 26
5.1.1.4.6 `substring`....26
5.1.1.4.7 `tolower` 26
5.1.1.4.8 `toupper` 27
5.1.1.4.9 `trim` 27
5.1.1.4.10 `concat` 27
5.1.1.4.11 `year` 27
5.1.1.4.12 `month` 28
5.1.1.4.13 `day` 28
5.1.1.4.14 `hour` 28
5.1.1.4.15 `minute` 29
5.1.1.4.16 `second` 29
5.1.1.4.17 `fractionalseconds` 29
5.1.1.4.18 `date` 29
5.1.1.4.19 `time` 29
5.1.1.4.20 `totaloffsetminutes` 30
5.1.1.4.21 `now` 30
5.1.1.4.22 `maxdatetime` 30
5.1.1.4.23 `mindatetime` 30
5.1.1.4.24 `totalseconds` 30
5.1.1.4.25 `round` 30
5.1.1.4.26 `floor` 31
5.1.1.4.27 `ceiling`....31
5.1.1.4.28 `isof` 31
5.1.1.4.29 `cast` 31
5.1.1.4.30 `geo.distance` 32
5.1.1.4.31 `geo.intersects`....32
5.1.1.4.32 `geo.length` 32
5.1.1.5 Lambda Operators 33
5.1.1.5.1 `any` 33
5.1.1.5.2 `all` 33
5.1.1.6 Literals 33
5.1.1.6.1 Primitive Literals....33
5.1.1.6.2 Complex and Collection Literals....33
5.1.1.6.3 `null` 34
5.1.1.6.4 `$it` 34
5.1.1.6.5 `$root` 34

5.1.1.7 Path Expressions 34
5.1.1.8 Parameter Aliases 35
5.1.1.9 Operator Precedence 35
5.1.1.10 Numeric Promotion 36
5.1.2 System Query Option `$expand` 36
5.1.3 System Query Option `$select` 38
5.1.4 System Query Option `$orderby` 40
5.1.5 System Query Options `$top` and `$skip` 40
5.1.6 System Query Option `$count` 40
5.1.7 System Query Option `$search` 40
5.1.7.1 Search Expressions 41
5.1.8 System Query Option `$format` 41
5.2 Custom Query Options 41
5.3 Parameter Aliases 41
6 Conformance 42
Appendix A. Acknowledgments 43
Appendix B. Revision History 44

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 a set of recommended (but not required) rules for constructing URLs to identify the data and metadata exposed by an OData service as well as a set of reserved URL query string operators, which if accepted by an OData service, MUST be implemented as required by this document.

The **[OData-Atom]** and **[OData-JSON]** documents specify the format of the resource representations that are exchanged using OData and the **[OData-Protocol]** document describes the actions that can be performed on the URLs (optionally constructed following the conventions defined in this document) embedded in those representations.

Services are encouraged to follow the URL construction conventions defined in this specification when possible as consistency promotes an ecosystem of reusable client components and libraries.

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 the 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-Protocol] *OData Version 4.0 Part 1: Protocol*.
See link in "Additional artifacts" section on cover page.

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

[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.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax”, STD 66, RFC 3986, January 2005. http://www.ietf.org/rfc/rfc3986.txt.

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

1.3 Typographical Conventions

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

```
Normative source code uses this paragraph style.
```

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

Example 1: text describing an example uses this paragraph style


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

All examples in this document are non-normative and informative only.

All other text is normative unless otherwise labeled.

2 URL Components

A URL used by an OData service has at most three significant parts: the *service root URL*, *resource path* and *query options*. Additional URL constructs (such as a fragment) can be present in a URL used by an OData service; however, this specification applies no further meaning to such additional constructs.

Example 2: OData URL broken down into its component parts:

```
http://host:port/path/SampleService.svc/Categories(1)/Products?$top=2&$orderby=Name
\_____________________________________/\__________________/ \__________________/
                  |                            |                     |
          service root URL              resource path          query options
```

Mandated and suggested content of these three significant URL components used by an OData service are covered in sequence in the three following chapters.

OData follows the URI syntax rules defined in **[RFC3986]** and in addition assigns special meaning to several of the sub-delimiters defined by **[RFC3986]**, so special care has to be taken regarding parsing and percent-decoding.

[RFC3986] defines three steps for URL processing that MUST be performed before percent-decoding:

- Split undecoded URL into components scheme, hier-part, query, and fragment at first "`:`", then first "`?`", and then first "`#`"
- Split undecoded hier-part into authority and path
- Split undecoded path into path segments at "`/`"

After applying these steps defined by RFC3986 the following steps MUST be performed:

- Split undecoded query at "`&`" into query options, and each query option at the first "`=`" into query option name and query option value
- Percent-decode path segments, query option names, and query option values exactly once
- Interpret path segments, query option names, and query option values according to OData rules

The OData rules are defined in this document and the **[OData-ABNF]**. Note that the rules in **[OData-ABNF]** assume that URIs and URI parts have been percent-encoding normalized as described in section 6.2.2.2 of **[RFC3986]** before applying the grammar to them, i.e. all characters in the unreserved set (see rule `unreserved` in **[OData-ABNF]**) are plain literals and not percent-encoded. For characters outside of the unreserved set that are significant to OData the ABNF rules explicitly state whether the percent-encoded representation is treated identical to the plain literal representation. This is done to make the input strings in the ABNF test cases more readable.

One of these rules is that single quotes within string literals are represented as two consecutive single quotes.

Example 3: valid OData URLs:

```
http://host/service/People('O''Neil')
http://host/service/People(%27O%27%27Neil%27)
http://host/service/People%28%27O%27%27Neil%27%29
http://host/service/Categories('Smartphone%2FTablet')
```

Example 4: invalid OData URLs:

```
http://host/service/People('O'Neil')
http://host/service/People('O%27Neil')
http://host/service/Categories('Smartphone/Tablet')
```

The first and second examples are invalid because a single quote in a string literal must be represented as two consecutive single quotes. The third example is invalid because forward slashes are interpreted as

path segment separators and `Categories('Smartphone` is not a valid OData path segment, nor is `Tablet')`.

3 Service Root URL

The service root URL identifies the root of an OData service. A `GET` request to this URL returns the format-specific service document, see **[OData-JSON]** and **[OData-Atom]**.

The service root URL always terminates in a forward slash.

The service document enables simple hypermedia-driven clients to enumerate and explore the resources published by the OData service.

4 Resource Path

The rules for resource path construction as defined in this section are optional. OData services SHOULD follow the subsequently described URL path construction rules and are indeed encouraged to do so; as such consistency promotes a rich ecosystem of reusable client components and libraries.

Services that do not follow the resource path conventions for entity container children are strongly encouraged to document their resource paths by annotating entity container children with the term `Core.ResourcePath` defined in **[OData-VocCore]**. The annotation value is the URL of the annotated resource and may be relative to `xml:base` (if present), otherwise the request URL.

Resources exposed by an OData service are addressable by corresponding resource path URL components to enable interaction of the client with that resource aspect.

To illustrate the concept, some examples for resources might be: customers, a single customer, orders related to a single customer, and so forth. Examples of addressable aspects of these resources as exposed by the data model might be: collections of entities, a single entity, properties, links, operations, and so on.

An OData service MAY respond with `301 Moved Permanently` or `307 Temporary Redirect` from the canonical URL to the actual URL.

4.1 Addressing the Model for a Service

OData services expose their entity model according to **[OData-CSDL]** at the metadata URL, formed by appending `$metadata` to the service root URL.

Example 5: Metadata document URL

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

OData services MAY expose their entity model as a service, according to **[OData-CSDL]**, by appending a trailing slash (`/`) to the metadata document URL.

Example 6: Metadata service root URL

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

4.2 Addressing the Batch Endpoint for a Service

OData services that support batch requests expose a batch URL formed by appending `$batch` to the service root URL.

Example 7: batch URL

```
http://host/service/$batch
```

4.3 Addressing Entities

The basic rules for addressing a collection (of entities), a single entity within a collection, a singleton, as well as a property of an entity are covered in the `resourcePath` syntax rule in **[OData-ABNF]**.

Below is a (non-normative) snippet from **[OData-ABNF]**:

```
resourcePath = entitySetName                    [collectionNavigation]
             / singleton                        [singleNavigation]
             / actionImportCall
             / entityColFunctionImportCall      [ collectionNavigation ]
             / entityFunctionImportCall         [ singleNavigation ]
             / complexColFunctionImportCall     [ collectionPath ]
             / complexFunctionImportCall        [ complexPath ]
             / primitiveColFunctionImportCall   [ collectionPath ]
             / primitiveFunctionImportCall      [ singlePath ]
             / crossjoin
             / '$all'
```

Since OData has a uniform composable URL syntax and associated rules there are many ways to address a collection of entities, including, but not limited to:

- Via an entity set (see rule `entitySetName` in **[OData-ABNF]**)

  Example 8:

  ```
  http://host/service/Products
  ```

- By navigating a collection-valued navigation property (see rule: `entityColNavigationProperty`)
- By invoking a function that returns a collection of entities (see rule: `entityColFunctionCall`)

  Example 9: function with parameters in resource path

  ```
  http://host/service/ProductsByCategoryId(categoryId=2)
  ```

  Example 10: function with parameters as query options

  ```
  http://host/service/ProductsByColor(color=@color)?@color='red'
  ```

- By invoking an action that returns a collection of entities (see rule: `actionCall`)

Likewise there are many ways to address a single entity.

Sometimes a single entity can be accessed directly, for example by:

- Invoking a function that returns a single entity (see rule: `entityFunctionCall`)
- Invoking an action that returns a single entity (see rule: `actionCall`)
- Addressing a singleton

  Example 11:

  ```
  http://host/service/BestProductEverCreated
  ```

Often however a single entity is accessed by composing more path segments to a `resourcePath` that identifies a collection of entities, for example by:

- Using an entity key to select a single entity (see rules: `collectionNavigation` and `keyPredicate`)

  Example 12:

  ```
  http://host/service/Categories(1)
  ```

- Invoking an action bound to a collection of entities that returns a single entity (see rule: `boundOperation`)
- Invoking an function bound to a collection of entities that returns a single entity (see rule: `boundOperation`)

  Example 13:


```
http://host/service/Products/Model.MostExpensive()
```

These rules are recursive, so it is possible to address a single entity via another single entity, a collection via a single entity and even a collection via a collection; examples include, but are not limited to:

- By following a navigation from a single entity to another related entity (see rule: `entityNavigationProperty`)

  Example 14:

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

- By invoking a function bound to a single entity that returns a single entity (see rule: `boundOperation`)

  Example 15:

  ```
  http://host/service/Products(1)/Model.MostRecentOrder()
  ```

- By invoking an action bound to a single entity that returns a single entity (see rule: `boundOperation`)
- By following a navigation from a single entity to a related collection of entities (see rule: `entityColNavigationProperty`)

  Example 16:

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

- By invoking a function bound to a single entity that returns a collection of entities (see rule: `boundOperation`)

  Example 17:

  ```
  http://host/service/Categories(1)/Model.TopTenProducts()
  ```

- By invoking an action bound to a single entity that returns a collection of entities (see rule: `boundOperation`)
- By invoking a function bound to a collection of entities that returns a collection of entities (see rule: `boundOperation`)

  Example 18:

  ```
  http://host/service/Categories(1)/Products/Model.AllOrders()
  ```

- By invoking an action bound to a collection of entities that returns a collection of entities (see rule: `boundOperation`)

Finally it is possible to compose path segments onto a resource path that identifies a primitive, complex instance, collection of primitives or collection of complex instances and bind an action or function that returns an entity or collections of entities.

4.3.1 Canonical URL

For OData services conformant with the addressing conventions in this section, the canonical form of an absolute URL identifying a non-contained entity is formed by adding a single path segment to the service root URL. The path segment is made up of the name of the entity set associated with the entity followed by the key predicate identifying the entity within the collection. No type-cast segment is added to the canonical URL, even if the entity is an instance of a type derived from the declared entity type of its entity set.

Example 19: Non-canonical URL


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

Example 20: Canonical URL for previous example:

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

4.3.2 Canonical URL for Contained Entities

For contained entities (i.e. related via a navigation property that specifies `ContainsTarget="true"`, see **[OData-CSDL]**) the canonical URL is the canonical URL of the containing entity followed by:

- A cast segment if the navigation property is defined on a type derived from the entity type declared for the entity set,
- A path segment for the containment navigation property, and
- If the navigation property returns a collection, a key predicate that uniquely identifies the entity in that collection.

4.3.3 URLs for Related Entities with Referential Constraints

If a navigation property leading to a related entity type has a partner navigation property that specifies a referential constraint, then those key properties of the related entity that take part in the referential constraint MAY be omitted from URLs.

Example 21: full key predicate of related entity

```
https://host/service/Orders(1)/Items(OrderID=1,ItemNo=2)
```

Example 22: shortened key predicate of related entity

```
https://host/service/Orders(1)/Items(2)
```

The two above examples are equivalent if the navigation property `Items` from `Order` to `OrderItem` has a partner navigation property from `OrderItem` to `Order` with a referential constraint tying the value of the `OrderID` key property of the `OrderItem` to the value of the `ID` property of the `Order`.

The shorter form that does not specify the constrained key parts redundantly is preferred. If the value of the constrained key is redundantly specified then it MUST match the principal key value.

4.3.4 Resolving an Entity-Id

To resolve an entity-id into a representation of the identified entity, the client issues a `GET` request to the `$entity` resource located at the URL `$entity` relative to the service root URL. The entity-id MUST be specified using the system query option `$id`. The entity-id may be expressed as an absolute IRI or relative to the service root URL.

Example 23: request the entity representation for an entity-id

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

The semantics of `$entity` are covered in the **[OData-Protocol]** document.

4.4 Addressing References between Entities

OData services are based on a data model that supports relationships as first class constructs. For example, an OData service could expose a collection of Products entities each of which are related to a Category entity.

References between entities are addressable in OData just like entities themselves are (as described above) by appending a navigation property name followed by `/$ref` to the entity URL.

Example 24: URL addressing the references between `Categories(1)` and `Products`


```
http://host/service/Categories(1)/Products/$ref
```

Resource paths addressing a single entity reference can be used in `DELETE` requests to unrelated two entities. Resource paths addressing collection of references can be used in `DELETE` requests if they are followed by the system query option `$id` identifying one of the entity references in the collection. The entity-id specified by `$id` may be expressed absolute or relative to the request URL. For details see **[OData-Protocol]**.

Example 25: two ways of unrelating `Categories(1)` and `Products(0)`

```
DELETE http://host/service/Categories(1)/Products/$ref?$id=../../Products(0)
DELETE http://host/service/Products(0)/Category/$ref
```

4.5 Addressing Operations

4.5.1 Addressing Actions

The semantic rules for addressing and invoking actions are defined in the **[OData-Protocol]** document. The grammar for addressing and invoking actions is defined by the following syntax grammar rules in **[OData-ABNF]**:

- The `actionImportCall` syntax rule defines the grammar in the `resourcePath` for addressing and invoking an action import directly from the service root.
- The `boundActionCall` syntax rule defines the grammar in the `resourcePath` for addressing and invoking an action that is appended to a `resourcePath` that identifies some resources that can be used as the binding parameter value when invoking the action.
- The `boundOperation` syntax rule (which encompasses the `boundActionCall` syntax rule), when used by the `resourcePath` syntax rule, illustrates how a `boundActionCall` can be appended to a `resourcePath`.

4.5.2 Addressing Functions

The semantic rules for addressing and invoking functions are defined in the **[OData-Protocol]** document. The grammar for addressing and invoking functions is defined by a number syntax grammar rules in **[OData-ABNF]**, in particular:

- The function import call syntax rules `complexFunctionImportCall`, `complexColFunctionImportCall`, `entityFunctionImportCall`, `entityColFunctionImportCall`, `primitiveFunctionImportCall`, and `primitiveColFunctionImportCall` define the grammar in the `resourcePath` for addressing and providing parameters for a function import directly from the service root.
- The bound function call syntax rules `boundComplexFunctionCall`, `boundComplexColFunctionCall`, `boundEntityFunctionCall`, `boundEntityColFunctionCall`, `boundPrimitiveFunctionCall` and `boundPrimitiveColFunctionCall` define the grammar in the `resourcePath` for addressing and providing parameters for a function that is appended to a `resourcePath` that identifies some resources that can be used as the binding parameter value when invoking the function.
- The `boundOperation` syntax rule (which encompasses the bound function call syntax rules), when used by the `resourcePath` syntax rule, illustrates how a bound function call can be appended to a `resourcePath`.
- The `functionExpr` and `boundFunctionExpr` syntax rules as used by the `filter` and `orderby` syntax rules define the grammar for invoking functions to help filter and order resources identified by the `resourcePath` of the URL.

- The `aliasAndValue` syntax rule defines the grammar for providing function parameter values using Parameter Alias Syntax, see **[OData-Protocol]**.

4.6 Addressing a Property

To address an entity property clients append a path segment containing the property name to the URL of the entity. If the property has a complex type value, properties of that value can be addressed by further property name composition.

4.7 Addressing a Property Value

To address the raw value of a primitive property, clients append a path segment containing the string `$value` to the property URL.

Properties of type `Edm.Stream` already return the raw value of the media stream and do not support appending the `$value` segment.

4.8 Addressing the Count of a Collection

To address the raw value of the number of items in a collection, clients append `/$count` to the resource path of the URL identifying the entity set or collection. The count is calculated after applying any `$filter` or `$search` system query options to the collection. The returned count MUST NOT be affected by `$top`, `$skip`, `$orderby`, or `$expand`.

Example 26: the number of related entities

```
http://host/service/Categories(1)/Products/$count
```

Example 27: the number of entities in an entity set

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

Example 28: entity count in a `$filter` expression. Note that the spaces around `gt` are for readability of the example only; in real URLs they must be percent-encoded as `%20`.

```
http://host/service/Categories?$filter=Products/$count gt 0
```

Example 29: entity count in an `$orderby` expression

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

4.9 Addressing Derived Types

Any resource path or path expression identifying a collection of entities or complex type instances can be appended with a path segment containing the qualified name of a type derived from the declared type of the collection. The result will be restricted to instances of the derived type and may be empty.

Any resource path or path expression identifying a single entity or complex type instance can be appended with a path segment containing the qualified name of a type derived from the declared type of the identified resource. If used in a resource path and the identified resource is not an instance of the derived type, the request will result in a `404 Not Found` response. If used in a path expression that is part of a Boolean expression, the type cast will evaluate to `null`.

Example 30: entity set restricted to `VipCustomer` instances

```
http://host/service/Customers/Model.VipCustomer
```

Example 31: entity restricted to a `VipCustomer` instance, resulting in `404 Not Found` if the customer with key `1` is not a `VipCustomer`

```
http://host/service/Customers/Model.VipCustomer(1)
http://host/service/Customers(1)/Model.VipCustomer
```

Example 32: cast the complex property `Address` to its derived type `DetailedAddress`, then get a property of the derived type

```
http://host/service/Customers(1)/Address/Model.DetailedAddress/Location
```

Example 33: filter expression with type cast; will evaluate to `null` for all non-`VipCustomer` instances and thus return only instances of `VipCustomer`

```
http://host/service/Customers?
       $filter=Model.VipCustomer/PercentageOfVipPromotionProductsOrdered gt 80
```

Example 34: expand the single related `Customer` only if it is an instance of `Model.VipCustomer`. For to-many relationships only `Model.VipCustomer` instances would be inlined,

```
http://host/service/Orders?$expand=Customer/Model.VipCustomer
```

4.10 Addressing the Media Stream of a Media Entity

To address the media stream represented by a media entity, clients append `/$value` to the resource path of the media entity URL. Services may redirect from this canonical URL to the source URL of the media stream.

Example 35: request the media stream for the picture with the key value `Sunset4321299432`:

```
http://host/service/Pictures('Sunset4321299432')/$value
```

4.11 Addressing the Cross Join of Entity Sets

In addition to querying related entities through navigation properties defined in the entity model of a service, the cross join operator allows querying across unrelated entity sets.

The cross join is addressed by appending the path segment `$crossjoin` to the service root URL, followed by the parenthesized comma-separated list of joined entity sets. It returns the Cartesian product of all the specified entity sets, represented as a collection of instances of a virtual complex type. Each instance consists of one non-nullable, single-valued navigation property per joined entity set. Each such navigation property is named identical to the corresponding entity set, with a target type equal to the declared entity type of the corresponding entity set.

The `$filter` and `$orderby` query options can be specified using properties of the entities in the selected entity sets, prepended with the entity set as the navigation property name.

The `$expand` query option can be specified using the names of the selected entity sets as navigation property names. If a selected entity set is not expanded, it MUST be represented using the read URL of the related entity as a navigation link in the complex type instance.

The `$count`, `$skip`, and `$top` query options can also be used with no special semantics.

Example 36: if `Sales` had a structural property `ProductID` instead of a navigation property `Product`, a "cross join" between `Sales` and `Products` could be addressed

```
http://host/service/$crossjoin(Products,Sales)?
                                    $filter=Products/ID eq Sales/ProductID
```

and would result in


```
{

"@odata.context":"http://host/service/$metadata#Collection(Edm.ComplexType)",
  "value":[
    {
      "Products@odata.navigationLink":"Products(0)",
      "Sales@odata.navigationLink":"Sales(42)",
    },
    {
      "Products@odata.navigationLink":"Products(0)",
      "Sales@odata.navigationLink":"Sales(57)",
    },
    ...
    {
      "Products@odata.navigationLink":"Products(99)",
      "Sales@odata.navigationLink":"Sales(21)",
    }
  ]
}
```

4.12 Addressing All Entities in a Service

The symbolic resource `$all`, located at the service root, identifies the collection of all entities in a service, i.e. the union of all entity sets plus all singletons.

This symbolic resource is of type `Collection(Edm.EntityType)` and allows the `$search` system query option plus all other query options applicable to collections of entities.

The `$all` resource can be appended with a path segment containing the qualified name of an entity type in order to restrict the collections to entities of that type. Query options such as `$select`, `$filter`, `$expand` and `$orderby` can be applied to this restricted set according to the specified type.

Example 37: all entities in a service that somehow match `red`

```
http://host/service/$all?$search=red
```

Example 38: all `Customer` entities in a service whose name contains `red`

```
http://host/service/$all/Model.Customer?$filter=contains(Name,'red')
```


5 Query Options

The query options part of an OData URL specifies three types of information: system query options, custom query options, and parameter aliases. All OData services MUST follow the query string parsing and construction rules defined in this section and its subsections.

5.1 System Query Options

System query options are query string parameters that control the amount and order of the data returned for the resource identified by the URL. The names of all system query options are prefixed with a dollar (`$`) character.

For `GET` requests the following rules apply:

- Resource paths identifying a single entity, a complex type instance, a collection of entities, or a collection of complex type instances allow `$expand` and `$select`.
- Resource paths identifying a collection allow `$filter`, `$count`, `$orderby`, `$skip`, and `$top`.
- Resource paths identifying a collection of entities allow `$search`.
- Resource paths ending in `/$count` allow `$filter` and `$search`.
- Resource paths not ending in `/$count` or `/$batch` allow `$format`.

For `POST` requests to an action URL the return type of the action determines the applicable system query options that a service MAY support, following the same rules as `GET` requests.

`POST` requests to entity sets as well as all `PUT` and `DELETE` requests do not allow system query options.

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 reject any request that contains the unsupported option.

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

The semantics of all system query options are defined in the **[OData-Protocol]** document.

The grammar and syntax rules for system query options are defined in **[OData-ABNF]**.

Dynamic properties can be used in the same way as declared properties. If they are not defined on an instance, they evaluate to `null`.

5.1.1 System Query Option `$filter`

The `$filter` system query option allows clients to filter a collection of resources that are addressed by a request URL. The expression specified with `$filter` is evaluated for each resource in the collection, and only items where the expression evaluates to true are included in the response. Resources for which the expression evaluates to false or to null, or which reference properties that are unavailable due to permissions, are omitted from the response.

The **[OData-ABNF]** `filter` syntax rule defines the formal grammar of the `$filter` query option.

5.1.1.1 Logical Operators

OData defines a set of logical operators that evaluate to true or false (i.e. a `boolCommonExpr` as defined in **[OData-ABNF]**). Logical operators are typically used to filter a collection of resources.

Operands of collection, entity, and complex types are not supported in logical operators.

The syntax rules for the logical operators are defined in **[OData-ABNF]**.

The six comparison operators can be used with all primitive values except `Edm.Binary`, `Edm.Stream`, and the `Edm.Geo` types. `Edm.Binary`, `Edm.Stream`, and the `Edm.Geo` types can only be compared to the `null` value using the `eq` and `ne` operators.

5.1.1.1.1 Equals

The `eq` operator returns true if the left operand is equal to the right operand, otherwise it returns false.

The `null` value is equal to itself, and only to itself.

5.1.1.1.2 Not Equals

The `ne` operator returns true if the left operand is not equal to the right operand, otherwise it returns false.

The `null` value is not equal to any value but itself.

5.1.1.1.3 Greater Than

The `gt` operator returns true if the left operand is greater than the right operand, otherwise it returns false.

If any operand is `null`, the operator returns false.

For Boolean values true is greater than false.

5.1.1.1.4 Greater Than or Equal

The `ge` operator returns true if the left operand is greater than or equal to the right operand, otherwise it returns false.

If only one operand is `null`, the operator returns false. If both operands are `null`, it returns true because `null` is equal to itself.

5.1.1.1.5 Less Than

The `lt` operator returns true if the left operand is less than the right operand, otherwise it returns false.

If any operand is `null`, the operator returns false.

5.1.1.1.6 Less Than or Equal

The `le` operator returns true if the left operand is less than or equal to the right operand, otherwise it returns false.

If only one operand is `null`, the operator returns false. If both operands are `null`, it returns true because `null` is equal to itself.

5.1.1.1.7 And

The `and` operator returns true if both the left and right operands evaluate to true, otherwise it returns false.

The `null` value is treated as unknown, so if one operand evaluates to `null` and the other operand to false, the `and` operator returns false. All other combinations with `null` return `null`.

5.1.1.1.8 Or

The `or` operator returns false if both the left and right operands both evaluate to false, otherwise it returns true.

The `null` value is treated as unknown, so if one operand evaluates to `null` and the other operand to true, the `or` operator returns true. All other combinations with `null` return `null`.

5.1.1.1.9 Not

The `not` operator returns true if the operand returns false, otherwise it returns false.

The `null` value is treated as unknown, so `not null` returns `null`.

5.1.1.1.10 has

The has operator returns true if the right hand operand is an enumeration value whose flag(s) are set on the left operand.

The null value is treated as unknown, so if one operand evaluates to null, the has operator returns null.

5.1.1.1.11 Logical Operator Examples

The following examples illustrate the use and semantics of each of the logical operators.

Example 39: all products with a Name equal to 'Milk'

```
http://host/service/Products?$filter=Name eq 'Milk'
```

Example 40: all products with a Name not equal to 'Milk'

```
http://host/service/Products?$filter=Name ne 'Milk'
```

Example 41: all products with a Name greater than 'Milk':

```
http://host/service/Products?$filter=Name gt 'Milk'
```

Example 42: all products with a Name greater than or equal to 'Milk':

```
http://host/service/Products?$filter=Name ge 'Milk'
```

Example 43: all products with a Name less than 'Milk':

```
http://host/service/Products?$filter=Name lt 'Milk'
```

Example 44: all products with a Name less than or equal to 'Milk':

```
http://host/service/Products?$filter=Name le 'Milk'
```

Example 45: all products with the Name 'Milk' that also have a Price less than 2.55:

```
http://host/service/Products?$filter=Name eq 'Milk' and Price lt 2.55
```

Example 46: all products that either have the Name 'Milk' or have a Price less than 2.55:

```
http://host/service/Products?$filter=Name eq 'Milk' or Price lt 2.55
```

Example 47: all products that do not have a Name that ends with 'ilk':

```
http://host/service/Products?$filter=not endswith(Name,'ilk')
```

Example 48: all products whose style value includes Yellow:

```
http://host/service/Products?$filter=style has Sales.Pattern'Yellow'
```

5.1.1.2 Arithmetic Operators

OData defines a set of arithmetic operators that require operands that evaluate to numeric types. Arithmetic operators are typically used to filter a collection of resources. However services MAY allow using arithmetic operators with the $orderby system query option.

If an operand of an arithmetic operator is null, the result is null.

The syntax rules for the arithmetic operators are defined in **[OData-ABNF]**.

5.1.1.2.1 Addition

The add operator adds the left and right numeric operands.

The `add` operator is also valid for the following time-related operands:

- `DateTimeOffset add Duration` results in a `DateTimeOffset`
- `Duration add Duration` results in a `Duration`
- `Date add Duration` results in a `DateTimeOffset`

5.1.1.2.2 Subtraction

The `sub` operator subtracts the right numeric operand from the left numeric operand.

The `sub` operator is also valid for the following time-related operands:

- DateTimeOffset sub Duration results in a `DateTimeOffset`
- `Duration sub Duration` results in a `Duration`
- `DateTimeOffset sub DateTimeOffset` results in a `Duration`
- `Date sub Duration` results in a `DateTimeOffset`
- `Date sub Date` results in a `Duration`

5.1.1.2.3 Negation

The negation operator, represented by a minus (`-`) sign, changes the sign of its numeric or `Duration` operand.

5.1.1.2.4 Multiplication

The `mul` operator multiplies the left and right numeric operands.

5.1.1.2.5 Division

The `div` operator divides the left numeric operand by the right numeric operand. If the right operand is zero and the left operand is neither of type `Edm.Single` nor `Edm.Double`, the request fails. If the left operand is of type `Edm.Single` or `Edm.Double`, then positive `div` zero returns `INF`, negative `div` zero returns `-INF`, and zero `div` zero returns `NaN`.

5.1.1.2.6 Modulo

The `mod` operator returns the remainder when the left integral operand is divided by the right integral operand. If the right operand is negative, the sign of the result is the same as the sign of the left operand. If the right operand is zero, the request fails.

5.1.1.2.7 Arithmetic Operator Examples

The following examples illustrate the use and semantics of each of the Arithmetic operators.

Example 49: all products with a Price of 2.55:

```
http://host/service/Products?$filter=Price add 2.45 eq 5.00
```

Example 50: all products with a Price of 2.55:

```
http://host/service/Products?$filter=Price sub 0.55 eq 2.00
```

Example 51: all products with a Price of 2.55:

```
http://host/service/Products?$filter=Price mul 2.0 eq 5.10
```

Example 52: all products with a Price of 2.55:

```
http://host/service/Products?$filter=Price div 2.55 eq 1
```


Example 53: all products with a Rating exactly divisible by 5:

```
http://host/service/Products?$filter=Rating mod 5 eq 0
```

5.1.1.3 Grouping

The Grouping operator (open and close parenthesis "`( )`") controls the evaluation order of an expression. The Grouping operator returns the expression grouped inside the parenthesis.

Example 54: all products because 9 mod 3 is 0

```
http://host/service/Products?$filter=(4 add 5) mod (4 sub 1) eq 0
```

5.1.1.4 Canonical Functions

In addition to operators, a set of functions is also defined for use with the `$filter` or `$orderby` system query options. The following sections describe the available functions. Note: ISNULL or COALESCE operators are not defined. Instead, OData defines a `null` literal that can be used in comparisons.

If a parameter of a canonical function is `null`, the function returns `null`.

The syntax rules for all functions are defined in **[OData-ABNF]**.

5.1.1.4.1 `contains`

The `contains` function has the following signature:

```
Edm.Boolean contains(Edm.String,Edm.String)
```

The `contains` function returns `true` if the second parameter string value is a substring of the first parameter string value, otherwise it returns `false`. The `containsMethodCallExpr` syntax rule defines how the `contains` function is invoked.

Example 55: all customers with a `CompanyName` that contains `'Alfreds'`

```
http://host/service/Customers?$filter=contains(CompanyName,'Alfreds')
```

5.1.1.4.2 `endswith`

The `endswith` function has the following signature:

```
Edm.Boolean endswith(Edm.String,Edm.String)
```

The `endswith` function returns `true` if the first parameter string value ends with the second parameter string value, otherwise it returns `false`. The `endsWithMethodCallExpr` syntax rule defines how the `endswith` function is invoked.

Example 56: all customers with a `CompanyName` that ends with `'Futterkiste'`

```
http://host/service/Customers?$filter=endswith(CompanyName,'Futterkiste')
```

5.1.1.4.3 `startswith`

The `startswith` function has the following signature:

```
Edm.Boolean startswith(Edm.String,Edm.String)
```

The `startswith` function returns `true` if the first parameter string value starts with the second parameter string value, otherwise it returns `false`. The `startsWithMethodCallExpr` syntax rule defines how the `startswith` function is invoked.

Example 57: all customers with a `CompanyName` that starts with `'Alfr'`


```
http://host/service/Customers?$filter=startswith(CompanyName,'Alfr')
```

5.1.1.4.4 length

The `length` function has the following signature:

```
Edm.Int32 length(Edm.String)
```

The `length` function returns the number of characters in the parameter value. The `lengthMethodCallExpr` syntax rule defines how the `length` function is invoked.

Example 58: all customers with a `CompanyName` that is 19 characters long

```
http://host/service/Customers?$filter=length(CompanyName) eq 19
```

5.1.1.4.5 indexof

The `indexof` function has the following signature:

```
Edm.Int32 indexof(Edm.String,Edm.String)
```

The `indexof` function returns the zero-based character position of the first occurrence of the second parameter value in the first parameter value. The `indexOfMethodCallExpr` syntax rule defines how the `indexof` function is invoked.

Example 59: all customers with a `CompanyName` containing `'lfreds'` starting at the second character

```
http://host/service/Customers?$filter=indexof(CompanyName,'lfreds') eq 1
```

5.1.1.4.6 substring

The `substring` function has consists of two overloads, with the following signatures:

```
Edm.String substring(Edm.String,Edm.Int32)
Edm.String substring(Edm.String,Edm.Int32,Edm.Int32)
```

The two argument `substring` function returns a substring of the first parameter string value, starting at the Nth character and finishing at the last character (where N is the second parameter integer value). The three argument `substring` function returns a substring of the first parameter string value identified by selecting M characters starting at the Nth character (where N is the second parameter integer value and M is the third parameter integer value).

The `substringMethodCallExpr` syntax rule defines how the `substring` functions are invoked.

Example 60: all customers with a `CompanyName` of `'lfreds Futterkiste'` once the first character has been removed

```
http://host/service/Customers?
                    $filter=substring(CompanyName, 1) eq 'lfreds Futterkiste'
```

Example 61: all customers with a `CompanyName` that has `'lf'` as the second and third characters

```
http://host/service/Customers?$filter=substring(CompanyName,1,2) eq 'lf'
```

5.1.1.4.7 tolower

The `tolower` function has the following signature:

```
Edm.String tolower(Edm.String)
```

The `tolower` function returns the input parameter string value with all uppercase characters converted to lowercase according to Unicode rules. The `toLowerMethodCallExpr` syntax rule defines how the `tolower` function is invoked.

Example 62: all customers with a `CompanyName` that equals `'alfreds futterkiste'` once any uppercase characters have been converted to lowercase

```
http://host/service/Customers?
                          $filter=tolower(CompanyName) eq 'alfreds futterkiste'
```

5.1.1.4.8 toupper

The `toupper` function has the following signature:

```
Edm.String toupper(Edm.String)
```

The `toupper` function returns the input parameter string value with all lowercase characters converted to uppercase according to Unicode rules. The `toUpperMethodCallExpr` syntax rule defines how the `toupper` function is invoked.

Example 63: all customers with a `CompanyName` that equals `'ALFREDS FUTTERKISTE'` once any lowercase characters have been converted to uppercase

```
http://host/service/Customers?
                          $filter=toupper(CompanyName) eq 'ALFREDS FUTTERKISTE'
```

5.1.1.4.9 trim

The `trim` function has the following signature:

```
Edm.String trim(Edm.String)
```

The `trim` function returns the input parameter string value with all leading and trailing whitespace characters, according to Unicode rules, removed. The `trimMethodCallExpr` syntax rule defines how the `trim` function is invoked.

Example 64: all customers with a `CompanyName` without leading or trailing whitespace characters

```
http://host/service/Customers?$filter=trim(CompanyName) eq CompanyName
```

5.1.1.4.10 concat

The `concat` function has the following signature:

```
Edm.String concat(Edm.String,Edm.String)
```

The `concat` function returns a string that appends the second input parameter string value to the first. The `concatMethodCallExpr` syntax rule defines how the `concat` function is invoked.

Example 65: all customers from Berlin, Germany

```
http://host/service/Customers?
                $filter=concat(concat(City,', '), Country) eq 'Berlin, Germany'
```

5.1.1.4.11 year

The `year` function has the following signatures:

```
Edm.Int32 year(Edm.Date)
Edm.Int32 year(Edm.DateTimeOffset)
```

The `year` function returns the year component of the `Date` or `DateTimeOffset` parameter value, evaluated in the time zone of the `DateTimeOffset` parameter value. The `yearMethodCallExpr` syntax rule defines how the `year` function is invoked.

Services that are unable to preserve the offset of `Edm.DateTimeOffset` values and instead normalize the values to some common time zone (i.e. UTC) MUST fail evaluation of the `year` function for literal `Edm.DateTimeOffset` values that are not stated in the time zone of the normalized values.

Example 66: all employees born in 1971

```
http://host/service/Employees?$filter=year(BirthDate) eq 1971
```

5.1.1.4.12 month

The `month` function has the following signatures:

```
Edm.Int32 month(Edm.Date)
Edm.Int32 month(Edm.DateTimeOffset)
```

The `month` function returns the month component of the `Date` or `DateTimeOffset` parameter value, evaluated in the time zone of the `DateTimeOffset` parameter value. The `monthMethodCallExpr` syntax rule defines how the `month` function is invoked.

Services that are unable to preserve the offset of `Edm.DateTimeOffset` values and instead normalize the values to some common time zone (i.e. UTC) MUST fail evaluation of the `month` function for literal `Edm.DateTimeOffset` values that are not stated in the time zone of the normalized values.

Example 67: all employees born in May

```
http://host/service/Employees?$filter=month(BirthDate) eq 5
```

5.1.1.4.13 day

The `day` function has the following signatures:

```
Edm.Int32 day(Edm.Date)
Edm.Int32 day(Edm.DateTimeOffset)
```

The `day` function returns the day component `Date` or `DateTimeOffset` parameter value, evaluated in the time zone of the `DateTimeOffset` parameter value. The `dayMethodCallExpr` syntax rule defines how the `day` function is invoked.

Services that are unable to preserve the offset of `Edm.DateTimeOffset` values and instead normalize the values to some common time zone (i.e. UTC) MUST fail evaluation of the `day` function for literal `Edm.DateTimeOffset` values that are not stated in the time zone of the normalized values.

Example 68: all employees born on the 8th day of a month

```
http://host/service/Employees?$filter=day(BirthDate) eq 8
```

5.1.1.4.14 hour

The `hour` function has the following signatures:

```
Edm.Int32 hour(Edm.DateTimeOffset)
Edm.Int32 hour(Edm.TimeOfDay)
```

The `hour` function returns the hour component of the `DateTimeOffset` or `TimeOfDay` parameter value, evaluated in the time zone of the `DateTimeOffset` parameter value. The `hourMethodCallExpr` syntax rule defines how the `hour` function is invoked.

Services that are unable to preserve the offset of `Edm.DateTimeOffset` values and instead normalize the values to some common time zone (i.e. UTC) MUST fail evaluation of the `hour` function for literal `Edm.DateTimeOffset` values that are not stated in the time zone of the normalized values.

Example 69: all employees born in the 4th hour of a day


```
http://host/service/Employees?$filter=hour(BirthDate) eq 4
```

5.1.1.4.15 minute

The `minute` function has the following signatures:

```
Edm.Int32 minute(Edm.DateTimeOffset)
Edm.Int32 minute(Edm.TimeOfDay)
```

The `minute` function returns the minute component of the `DateTimeOffset` or `TimeOfDay` parameter value, evaluated in the time zone of the `DateTimeOffset` parameter value. The `minuteMethodCallExpr` syntax rule defines how the `minute` function is invoked.

Example 70: all employees born in the 40th minute of any hour on any day

```
http://host/service/Employees?$filter=minute(BirthDate) eq 40
```

5.1.1.4.16 second

The `second` function has the following signatures:

```
Edm.Int32 second(Edm.DateTimeOffset)
Edm.Int32 second(Edm.TimeOfDay)
```

The `second` function returns the second component (without the fractional part) of the `DateTimeOffset` or `TimeOfDay` parameter value. The `secondMethodCallExpr` syntax rule defines how the `second` function is invoked.

Example 71: all employees born in the 40th second of any minute of any hour on any day

```
http://host/service/Employees?$filter=second(BirthDate) eq 40
```

5.1.1.4.17 fractionalseconds

The `fractionalseconds` function has the following signatures:

```
Edm.Decimal fractionalseconds(Edm.DateTimeOffset)
Edm.Decimal fractionalseconds(Edm.TimeOfDay)
```

The `fractionalseconds` function returns the fractional seconds component of the `DateTimeOffset` or `TimeOfDay` parameter value as a non-negative decimal value less than 1. The `fractionalsecondsMethodCallExpr` syntax rule defines how the `fractionalseconds` function is invoked.

Example 72: all employees born less than 100 milliseconds after a full second of any minute of any hour on any day

```
http://host/service/Employees?$filter=fractionalseconds(BirthDate) lt 0.1
```

5.1.1.4.18 date

The `date` function has the following signature:

```
Edm.Date date(Edm.DateTimeOffset)
```

The `date` function returns the date part of the `DateTimeOffset` parameter value, evaluated in the time zone of the `DateTimeOffset` parameter value.

5.1.1.4.19 time

The `time` function has the following signature:

```
Edm.TimeOfDay time(Edm.DateTimeOffset)
```


The `time` function returns the time part of the `DateTimeOffset` parameter value, evaluated in the time zone of the `DateTimeOffset` parameter value.

Services that are unable to preserve the offset of `Edm.DateTimeOffset` values and instead normalize the values to some common time zone (i.e. UTC) MUST fail evaluation of the `time` function for literal `Edm.DateTimeOffset` values that are not stated in the time zone of the normalized values.

5.1.1.4.20 totaloffsetminutes

The `totaloffsetminutes` function has the following signature:

```
Edm.Int32 totaloffsetminutes(Edm.DateTimeOffset)
```

The `totaloffsetminutes` function returns the signed number of minutes in the time zone offset part of the `DateTimeOffset` parameter value, evaluated in the time zone of the `DateTimeOffset` parameter value.

5.1.1.4.21 now

The `now` function has the following signature:

```
Edm.DateTimeOffset now()
```

The `now` function returns the current point in time (date and time with time zone) as a `DateTimeOffset` value.

Services are free to choose the time zone for the current point, e.g. UTC. Services that are unable to preserve the offset of `Edm.DateTimeOffset` values and instead normalize the values to some common time zone SHOULD return a value in the normalized time zone (i.e., UTC).

5.1.1.4.22 maxdatetime

The `maxdatetime` function has the following signature:

```
Edm.DateTimeOffset maxdatetime()
```

The `maxdatetime` function returns the latest possible point in time as a `DateTimeOffset` value.

5.1.1.4.23 mindatetime

The `mindatetime` function has the following signature:

```
Edm.DateTimeOffset mindatetime()
```

The `mindatetime` function returns the earliest possible point in time as a `DateTimeOffset` value.

5.1.1.4.24 totalseconds

The `totalseconds` function has the following signature:

```
Edm.Decimal totalseconds(Edm.Duration)
```

The `totalseconds` function returns the duration of the value in total seconds, including fractional seconds.

5.1.1.4.25 round

The `round` function has the following signatures

```
Edm.Double round(Edm.Double)
Edm.Decimal round(Edm.Decimal)
```

The `round` function rounds the input numeric parameter to the nearest numeric value with no decimal component. The mid-point between two integers is rounded away from zero, i.e. 0.5 is rounded to 1 and

-0.5 is rounded to -1. The `roundMethodCallExpr` syntax rule defines how the `round` function is invoked.

Example 73: all orders with freight costs that round to 32

```
http://host/service/Orders?$filter=round(Freight) eq 32
```

5.1.1.4.26 floor

The `floor` function has the following signatures

```
Edm.Double floor(Edm.Double)
Edm.Decimal floor(Edm.Decimal)
```

The `floor` function rounds the input numeric parameter down to the nearest numeric value with no decimal component. The `floorMethodCallExpr` syntax rule defines how the `floor` function is invoked.

Example 74: all orders with freight costs that round down to 32

```
http://host/service/Orders?$filter=floor(Freight) eq 32
```

5.1.1.4.27 ceiling

The `ceiling` function has the following signatures

```
Edm.Double ceiling(Edm.Double)
Edm.Decimal ceiling(Edm.Decimal)
```

The `ceiling` function rounds the input numeric parameter up to the nearest numeric value with no decimal component. The `ceilingMethodCallExpr` syntax rule defines how the `ceiling` function is invoked.

Example 75: all orders with freight costs that round up to 32

```
http://host/service/Orders?$filter=ceiling(Freight) eq 32
```

5.1.1.4.28 isof

The `isof` function has the following signatures

```
Edm.Boolean isof(type)
Edm.Boolean isof(expression,type)
```

The single parameter `isof` function returns `true` if the current instance is assignable to the type specified, according to the assignment rules for the `cast` function, otherwise it returns `false`.

The two parameter `isof` function returns `true` if the object referred to by the expression is assignable to the type specified, according to the same rules, otherwise it returns `false`.

The `isofExpr` syntax rule defines how the `isof` function is invoked.

Example 76: orders that are also `BigOrders`

```
http://host/service/Orders?$filter=isof(NorthwindModel.BigOrder)
http://host/service/Orders?$filter=isof($it,NorthwindModel.BigOrder)
```

Example 77: orders of a customer that is a `VIPCustomer`

```
http://host/service/Orders?$filter=isof(Customer,NorthwindModel.VIPCustomer)
```

5.1.1.4.29 cast

The `cast` function has the following signatures:


```
type cast(type)
type cast(expression,type)
```

The single parameter `cast` function returns the current instance cast to the type specified. The two-parameter `cast` function returns the object referred to by the expression cast to the type specified.

The `cast` function follows these assignment rules:

- The `null` value can be cast to any type.
- Primitive types are cast to `Edm.String` or a type definition based on it by using the literal representation used in payloads, and WKT (well-known text) format for `Geo` types, see rules `fullCollectionLiteral`, `fullLineStringLiteral`, `fullMultiPointLiteral`, `fullMultiLineStringLiteral`, `fullMultiPolygonLiteral`, `fullPointLiteral`, and `fullPolygonLiteral` in **[OData-ABNF]**. The cast fails if the target type specifies an insufficient `MaxLength`.
- Numeric primitive types are cast to each other with appropriate rounding. The cast fails if the integer part doesn't fit into target type.
- `Edm.DateTimeOffset`, `Edm.Duration`, and `Edm.TimeOfDay` values can be cast to the same type with a different precision with appropriate rounding.
- Structured types are assignable to their type or a direct or indirect base type.
- Collections are cast item by item.
- Services MAY support structural casting of entities and complex type instances to a derived type, or arbitrary structured type, by assigning values of identically named properties and casting them recursively. The cast fails if one of the property-value casts fails or the target type contains non-nullable properties that have not been assigned a value.

The `cast` function is optional for primitive values (first four rules) and up-casts (fifth rule).

If the cast fails the `cast` function returns `null`.

5.1.1.4.30 `geo.distance`

The `geo.distance` function has the following signatures:

```
Edm.Double geo.distance(Edm.GeographyPoint,Edm.GeographyPoint)
Edm.Double geo.distance(Edm.GeometryPoint,Edm.GeometryPoint)
```

The `geo.distance` function returns the shortest distance between the two points in the coordinate reference system signified by the two points' SRIDs.

5.1.1.4.31 `geo.intersects`

The `geo.intersects` function has the following signatures:

```
Edm.Boolean geo.intersects(Edm.GeographyPoint,Edm.GeographyPolygon)
Edm.Boolean geo.intersects(Edm.GeometryPoint,Edm.GeometryPolygon)
```

The `geo.intersects` function returns `true` if the specified point lies within the interior or on the boundary of the specified polygon, otherwise it returns `false`.

5.1.1.4.32 `geo.length`

The `geo.length` function has the following signatures:

```
Edm.Double geo.length(Edm.GeographyLineString)
Edm.Double geo.length(Edm.GeometryLineString)
```

The `geo.length` function returns the total length of its line string parameter in the coordinate reference system signified by its SRID.

5.1.1.5 Lambda Operators

OData defines two operators that evaluate a Boolean expression on a collection. Both must be prepended with a navigation path that identifies a collection. The argument of a lambda operator is a lambda variable name followed by a colon (`:`) and a Boolean expression that uses the lambda variable name to refer to properties of the related entities identified by the navigation path.

5.1.1.5.1 `any`

The `any` operator applies a Boolean expression to each member of a collection and returns `true` if the expression is `true` for any member of the collection, otherwise it returns `false`. The `any` operator without an argument returns `true` if the collection is not empty.

Example 78: all `Orders` that have any `Items` with a `Quantity` greater than `100`

```
http://host/service/Orders?$filter=Items/any(d:d/Quantity gt 100)
```

5.1.1.5.2 `all`

The `all` operator applies a Boolean expression to each member of a collection and returns `true` if the expression is `true` for all members of the collection, otherwise it returns `false`.

Example 79: all `Orders` that have only `Items` with a `Quantity` greater than `100`

```
http://host/service/Orders?$filter=Items/all(d:d/Quantity gt 100)
```

5.1.1.6 Literals

5.1.1.6.1 Primitive Literals

Primitive literals can appear in the resource path as key property values, and in the query part, for example, as operands in `$filter` expressions. They are represented according to the `primitiveLiteral` rule in **[OData-ABNF]**.

Example 80: expressions using primitive literals

```
NullValue eq null
TrueValue eq true
FalseValue eq false
Custom.Base64UrlDecode(binary'T0RhdGE') eq 'OData'
IntegerValue lt -128
DoubleValue ge 0.31415926535897931e1
SingleValue eq INF
DecimalValue eq 34.95
StringValue eq 'Say Hello,then go'
DateValue eq 2012-12-03
DateTimeOffsetValue eq 2012-12-03T07:16:23Z
DurationValue eq duration'P12DT23H59M59.999999999999S'
TimeOfDayValue eq 07:59:59.999
GuidValue eq 01234567-89ab-cdef-0123-456789abcdef
Int64Value eq 0
ColorEnumValue eq Sales.Pattern'Yellow',
geo.distance(Location,geography'SRID=0;Point(142.1 64.1)')
```

5.1.1.6.2 Complex and Collection Literals

Complex literals and collection literals in URLs are represented as JSON objects and arrays according to the `arrayOrObject` rule in **[OData-ABNF]**. Such literals MUST NOT appear in the path portion of the URL but can be passed to bound functions and function imports in path segments by using parameter aliases.

Note that the special characters `{`, `}`, `[`, `]`, and `"` MUST be percent-encoded in URLs although some browsers will accept and pass them on unencoded.

Example 81: collection of string literals

```
http://host/service/ProductsByColor?colors=["red","green"]
```

5.1.1.6.3 `null`

The `null` literal can be used to compare a value to null, or to pass a null value to a function.

5.1.1.6.4 `$it`

The `$it` literal can be used in expressions to refer to the current instance of the collection identified by the resource path. It can be used to compare properties of related entities to properties of the current instance in expressions within lambda operators, for example in `$filter` and `$orderby` expressions on collections of primitive types, or in `$filter` expressions nested within `$expand`. It can also be used as a path prefix to invoke a bound function on the current instance within an expression.

Example 82: email addresses ending with `.com` assuming `EmailAddresses` is a collection of strings

```
http://host/service/Customers(1)/EmailAddresses?$filter=endswith($it,'.com')
```

Example 83: customers along with their orders that shipped to the same city as the customer's address. The nested filter expression is evaluated in the context of Orders; `$it` allows referring to values in the outer context of Customers.

```
http://host/service/Customers?
            $expand=Orders($filter=$it/Address/City eq ShipTo/City)
```

Example 84: products with at least 10 positive reviews. `Model.PositiveReviews` is a function bound to `Model.Product` returning a collection of reviews.

```
http://host/service/Products?$filter=$it/Model.PositiveReviews()/$count ge 10
```

5.1.1.6.5 `$root`

The `$root` literal can be used in expressions to refer to resources of the same service. It can be used as a single-valued expression or within complex or collection literals.

Example 85: all employees with the same last name as employee `A1235`

```
http://host/service/Employees?
                        $filter=LastName eq $root/Employees('A1245')/LastName
```

Example 86: products ordered by a set of customers, where the set of customers is passed as a JSON array containing the resource paths from `$root` to each customer.

```
http://host/service/ProductsOrderedBy(Customers=@c)?
                       @c=[$root/Customers('ALFKI'),$root/Customers('BLAUS')]
```

5.1.1.7 Path Expressions

Properties and navigation properties of the entity type of the set of resources that are addressed by the request URL can be used as operands or function parameters, as shown in the preceding examples.

Properties of complex properties can be used via the same syntax as in resource paths, i.e. by specifying the name of a complex property, followed by a forward slash (`/`) and the name of a property of the complex property, and so on,

Properties and navigation properties of entities related with a target cardinality 0..1 or 1 can be used by specifying the navigation property, followed by a forward slash (`/`) and the name of a property of the related entity, and so on.

If a complex property is `null`, or no entity is related (in case of target cardinality 0..1), its value, and the values of its components, are treated as `null`.

Example 87: similar behavior whether `HeadquarterAddress` is a nullable complex type or a nullable navigation property

```
Companies(1)/HeadquarterAddress/Street
```

To access properties of derived types, the property name MUST be prefixed with the qualified name of the derived type on which the property is defined, followed by a forward slash (`/`), see addressing derived types. If the current instance is not of the specified derived type, the path expression returns `null`.

5.1.1.8 Parameter Aliases

Parameter aliases can be used within `$filter` or `$orderby` in place of expressions that evaluate to a primitive value, a complex value, or a collection of primitive or complex values. Parameter names start with the at sign (`@`) and can be used in more than one place in the expression. The value for the parameter alias is supplied in a query option with the same name as the parameter.

Example 88:

```
http://host/service/Movies?$filter=contains(@word,Title)&@word='Black'
```

Example 89:

```
http://host/service/Movies?$filter=Title eq @title&@title='Wizard of Oz'
```

5.1.1.9 Operator Precedence

OData services MUST use the following operator precedence for supported operators when evaluating `$filter` and `$orderby` expressions. Operators are listed by category in order of precedence from highest to lowest. Operators in the same category have equal precedence:

Group	Operator	Description	ABNF Expression
Grouping	`( )`	Precedence grouping	`parenExpr` `boolParenExpr`
Primary	`/`	Navigation	`firstMemberExpr` `memberExpr`
	`has`	Enumeration Flags	`hasExpr`

	`xxx( )`	Method Call	`methodCallExpr`
			`boolMethodCallExpr`
			`functionExpr`

Unary	`-`	Negation	`negateExpr`
	`not`	Logical Negation	`notExpr`
	`cast( )`	Type Casting	`castExpr`

Group	Operator	Description	ABNF Expression
Multiplicative	mul	Multiplication	mulExpr
	div	Division	divExpr
	mod	Modulo	modExpr
Additive	add	Addition	addExpr
	sub	Subtraction	subExpr
Relational	gt	Greater Than	gtExpr
	ge	Greater than or Equal	geExpr
	lt	Less Than	ltExpr
	le	Less than or Equal	leExpr
	isof	Type Testing	isofExpr
Equality	eq	Equal	eqExpr
	ne	Not Equal	neExpr
Conditional AND	and	Logical And	andExpr
Conditional OR	or	Logical Or	orExpr

5.1.1.10 Numeric Promotion

Services MAY support numeric promotion for arithmetic operations or when comparing two operands of comparable types by applying the following rules, in order:

- If either operand is `Edm.Double`, the other operand is converted to type `Edm.Double`.
- Otherwise, if either operand `is Edm.Single`, the other operand is converted to type `Edm.Single`.
- Otherwise, if either operand is of type `Edm.Decimal`, the other operand is converted to `Edm.Decimal`.
- Otherwise, if either operand is `Edm.Int64`, the other operand is converted to type `Edm.Int64`.
- Otherwise, if either operand is `Edm.Int32`, the other operand is converted to type `Edm.Int32`.
- Otherwise, if either operand is `Edm.Int16`, the other operand is converted to type `Edm.Int16`.

Each of these promotions uses the same semantics as a `castExpression` to promote an operand to the target type.

If the result of an arithmetic operation does not fit into the type determined by the above rules, the next-wider type is used in the above order, with `Edm.Double` considered widest.

OData does not define an implicit conversion between string and numeric types.

5.1.2 System Query Option `$expand`

The `$expand` system query option specifies the related resources to be included in line with retrieved resources.

What follows is a (non-normative) snippet from **[OData-ABNF]** that describes the syntax of `$expand`:

```
expand            = '$expand' EQ expandItem *( COMMA expandItem )
```



```
expandItem        = STAR [ ref / OPEN levels CLOSE ]
                  / expandPath
                    [ ref   [ OPEN expandRefOption
                            *( SEMI expandRefOption   ) CLOSE ]
                    / count [ OPEN expandCountOption
                            *( SEMI expandCountOption ) CLOSE ]
                    /         OPEN expandOption
                            *( SEMI expandOption      ) CLOSE
                    ]
expandPath        = [ ( qualifiedEntityTypeName
                      / qualifiedComplexTypeName
                      ) "/" ]
                    *( ( complexProperty / complexColProperty ) "/"
                       [ qualifiedComplexTypeName "/" ] )
                    navigationProperty
                    [ "/" qualifiedEntityTypeName ]
expandCountOption = filter
                  / search
expandRefOption   = expandCountOption
                  / orderby
                  / skip
                  / top
                  / inlinecount
expandOption      = expandRefOption
                  / select
                  / expand
                  / levels
```

Each `expandItem` is evaluated relative to the entity containing the navigation property being expanded.

A type cast using the `qualifiedEntityTypeName` to a type containing the property is required in order to expand a navigation property defined on a derived type.

An arbitrary number of single- or collection-valued complex properties, optionally followed by a type cast, allow drilling into complex properties.

The `navigationProperty` segment MUST identify a navigation property defined on the entity type of the request, the derived entity type specified in the type cast, or the last complex type identified by the complex property path.

Example 90: expand a navigation property of an entity type

```
http://host/service/Products?$expand=Category
```

Example 91: expand a navigation property of a complex type

```
http://host/service/Customers?$expand=Addresses/Country
```

A navigation property MUST NOT appear in more than one `expandItem`.

Query options can be applied to the expanded navigation property by appending a semicolon-separated list of query options, enclosed in parentheses, to the navigation property name. Allowed system query options are `$filter`, `$select`, `$orderby`, `$skip`, `$top`, `$count`, `$search`, and `$expand`.

Example 92: all categories and for each category all related products with a discontinued date equal to `null`

```
http://host/service/Categories?
                           $expand=Products($filter=DiscontinuedDate eq null)
```


The `$count` segment can be appended to the navigation property name or type-cast segment following the navigation property name to return just the count of the related entities. The `$filter` and `$search` system query options can be used to limit the number or related entities included in the count.

To retrieve entity references instead of the related entities, append `/$ref` to the navigation property name or type-cast segment following a navigation property name.

Example 93: all categories and for each category the references of all related products

```
http://host/service/Categories?$expand=Products/$ref
```

Example 94: all categories and for each category the references of all related products of the derived type `Sales.PremierProduct`

```
http://host/service/Categories?$expand=Products/Sales.PremierProduct/$ref
```

Example 95: all categories and for each category the references of all related premier products with a current promotion equal to `null`

```
http://host/service/Categories?
   $expand=Products/Sales.PremierProduct/$ref($filter=CurrentPromotion eq null)
```

Cyclic navigation properties (whose target type is identical or can be cast to its source type) can be recursively expanded using the special `$levels` option. The value of the `$levels` option is either a positive integer to specify the number of levels to expand, or the literal string `max` to specify the maximum expansion level supported by that service.

Example 96: all employees with their manager, manager's manager, and manager's manager's manager

```
http://host/service/Employees?$expand=Model.Manager/DirectReports($levels=3)
```

It is also possible to expand all declared and dynamic navigation properties using a star (`*`). To retrieve references to all related entities use `*/$ref`, and to expand all related entities with a certain distance use the star operator with the `$levels` option. The star operator can be combined with explicitly named navigation properties, which take precedence over the star operator.

Example 97: expand `Supplier` and include references for all other related entities

```
http://host/service/Categories?$expand=*/$ref,Supplier
```

Example 98: expand all related entities and their related entities

```
http://host/service/Categories?$expand=*($levels=2)
```

5.1.3 System Query Option `$select`

The `$select` system query option allows clients to requests a specific set of properties for each entity or complex type.

The `$select` query option is often used in conjunction with the `$expand` system query option, to define the extent of the resource graph to return (`$expand`) and then specify a subset of properties for each resource in the graph (`$select`). Expanded navigation properties MUST be returned, even if they are not specified as a `selectItem`.

What follows is a (non-normative) snippet from **[OData-ABNF]** showing the syntax of `$select`:

```
select         = '$select' EQ selectItem *( COMMA selectItem )

selectItem     = STAR
               / allOperationsInSchema
               / [ ( qualifiedEntityTypeName
                   / qualifiedComplexTypeName
```



```
                    ) "/" ]
                  ( selectProperty
                  / qualifiedActionName
                  / qualifiedFunctionName
                  )

selectProperty = primitiveProperty
               / primitiveColProperty
               / navigationProperty
               / selectPath [ "/" selectProperty ]

selectPath     = ( complexProperty / complexColProperty )
                 [ "/" qualifiedComplexTypeName ]
```

The `$select` system query option is interpreted relative to the entity type or complex type of the resources identified by the resource path section of the URL. Each `selectItem` in the `$select` clause indicates that the response MUST include the declared or dynamic properties, actions and functions identified by that `selectItem`. The simplest form of a `selectItem` explicitly requests a property defined on the entity type of the resources identified by the resource path section of the URL.

Example 99: rating and release date of all products

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

It is also possible to request all declared and dynamic structural properties using a star (`*`).

Example 100: all structural properties of all products

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

If the `selectItem` is not defined for the type of the resource, and that type is defined as open, then the property is treated as null for all instances on which it is not defined.

If the `selectItem` is not defined for the type of the resource, and that type is not defined as open, then the request is considered malformed.

If the `selectItem` is a navigation property then the corresponding navigation link is represented in the response. If the navigation property also appears in an `$expand` query option then it is additionally represented as inline content. This inline content can itself be restricted with a nested `$select` query option, see section 5.1.1.10.

Example 101: name and description of all products, plus name of expanded category

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

The `selectItem` MUST be prefixed with a `qualifiedEntityTypeName` or `qualifiedComplexTypeName` in order to select a property defined on a type derived from the type of the resource segment.

A `selectItem` that is a complex type or collection of complex type can be followed by a forward slash, an optional type cast segment, and the name of a property of the complex type (and so on for nested complex types).

Example 102: the `AccountRepresentative` property of any supplier that is of the derived type `Namespace.PreferredSupplier`, together with the `Street` property of the complex property `Address`, and the `Location` property of the derived complex type `Namespace.AddressWithLocation`

```
http://host/service/Suppliers?
     $select=Namespace.PreferredSupplier/AccountRepresentative,
             Address/Street,
             Address/Namespace.AddressWithLocation/Location
```

Any structural property, non-expanded navigation property, or operation not requested as a `selectItem` (explicitly or via a star) SHOULD be omitted from the response.

If any `selectItem` (including a star) is specified, actions and functions SHOULD be omitted unless explicitly requested using a `qualifiedActionName`, a `qualifiedFunctionName` or the `allOperationsInSchema`.

If an action or function is requested as a `selectItem`, either explicitly by using a `qualifiedActionName` or `qualifiedFunctionName` cause, or implicitly by using `allOperationsInSchema`, then the service includes information about how to invoke that operation for each entity identified by the last path segment in the request URL for which the operation can be bound.

If an action or function is requested in a `selectItem` using a `qualifiedActionName` or a `qualifiedFunctionName` and that operation cannot be bound to the entities requested, the service MUST ignore the `selectItem`.

Example 103: the `ID` property, the `ActionName` action defined in `Model` and all actions and functions defined in the `Model2` for each product if those actions and functions can be bound to that product

```
http://host/service/Products?$select=ID,Model.ActionName,Model2.*
```

When multiple `selectItems` exist in a select clause, then the total set of properties, open properties, navigation properties, actions and functions to be returned is equal to the union of the set of those identified by each `selectItem`.

If a `selectItem` is a path expression requesting a component of a complex property and the complex property is `null` on an instance, then the component is treated as `null` as well.

5.1.4 System Query Option `$orderby`

The `$orderby` system query option allows clients to request resources in a particular order.

The semantics of `$orderby` are covered in the **[OData-Protocol]** document.

The **[OData-ABNF]** `orderby` syntax rule defines the formal grammar of the `$orderby` query option.

5.1.5 System Query Options `$top` and `$skip`

The `$top` system query option requests the number of items in the queried collection to be included in the result. The `$skip` query option requests the number of items in the queried collection that are to be skipped and not included in the result. A client can request a particular page of items by combining `$top` and `$skip`.

The semantics of `$top` and `$skip` are covered in the **[OData-Protocol]** document. The **[OData-ABNF]** `top` and `skip` syntax rules define the formal grammar of the `$top` and `$skip` query options respectively.

5.1.6 System Query Option `$count`

The `$count` system query option allows clients to request a count of the matching resources included with the resources in the response. The `$count` query option has a Boolean value of `true` or `false`.

The semantics of `$count` is covered in the **[OData-Protocol]** document.

5.1.7 System Query Option `$search`

The `$search` system query option allows clients to request entities matching a free-text search expression.

The `$search` query option can be applied to a URL representing a collection of entities to return all matching entities within the collection. Applying the `$search` query option to the `$all` resource requests all matching entities in the service.

If both `$search` and `$filter` are applied to the same request, the results include only those entities that match both criteria.

The **[OData-ABNF]** `search` syntax rule defines the formal grammar of the `$search` query option.

Example 104: all products that are blue or green. It is up to the service to decide what makes a product blue or green.

```
http://host/service/Products?$search=blue OR green
```

5.1.7.1 Search Expressions

Search expressions are used within the `$search` system query option to request entities matching the specified expression.

Terms can be any single word to be matched within the expression.

Terms enclosed in double-quotes comprise a *phrase*.

Each individual term or phrase comprises a Boolean expression that returns `true` if the term or phrase is matched, otherwise `false`. The semantics of what is considered a match is dependent upon the service.

Expressions enclosed in parenthesis comprise a *group expression*.

The search expression can contain any number of terms, phrases, or group expressions, along with the case-sensitive keywords `NOT`, `AND`, and `OR`, evaluated in that order.

Expressions prefaced with `NOT` evaluate to `true` if the expression is not matched, otherwise `false`.

Two expressions not enclosed in quotes and separated by a space are equivalent to the same two expressions separated by the `AND` keyword. Such expressions evaluate to `true` if both of the expressions evaluate to `true`, otherwise `false`.

Expressions separated by an `OR` evaluate to `true` if either of the expressions evaluate to `true`, otherwise `false`.

The **[OData-ABNF]** `searchExpr` syntax rule defines the formal grammar of the search expression.

5.1.8 System Query Option `$format`

The `$format` system query option allows clients to request a response in a particular format and is useful for clients without access to request headers for standard content-type negotiation. Where present `$format` takes precedence over standard content-type negotiation.

The semantics of `$format` is covered in the **[OData-Protocol]** document.

The **[OData-ABNF]** `format` syntax rule define the formal grammar of the `$format` query option.

5.2 Custom Query Options

Custom query options provide an extensible mechanism for service-specific information to be placed in a URL query string. A custom query option is any query option of the form shown by the rule `customQueryOption` in **[OData-ABNF]**.

Custom query options MUST NOT begin with a `$` or `@` character.

Example 105: service-specific custom query option `debug-mode`

```
http://host/service/Products?debug-mode=true
```

5.3 Parameter Aliases

Parameter aliases can be used in place of literal values in function parameters or within a `$filter` or `$orderby` expression.

Parameter aliases MUST start with an `@` character.

The semantics of parameter aliases are covered in **[OData-Protocol]**.

The **[OData-ABNF]** rule `aliasAndValue` defines the formal grammar for passing parameter aliases as query options.

6 Conformance

The conformance requirements for OData clients and services are described in **[OData-Protocol]**.

Appendix A. Acknowledgments

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

Appendix B. Revision History

Revision	Date	Editor	Changes Made
Working Draft 01	2012-08-22	Michael Pizzo	Translated Contribution to OASIS format/template
Committee Specification Draft 01	2013-04-26	Ralf Handl Michael Pizzo Martin Zurmuehl	Added Full-Text Search, modified expand syntax, expand options, crosstabs, enumerations 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	Ralf Handl Michael Pizzo Martin Zurmuehl	Described which query options are applicable to which resource types and HTTP methods Simplified URL syntax Extended expand with a STAR operator Added special resources for cross-service search, cross joins, resolution of entity-ids Described handling of null values, division by zero, and overflow in arithmetic operations Added filtering for collections of complex and primitive types
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	Accessing properties of derived types Examples for primitive literals Precedence of `has` operator
Committee Specification 02	2013-11-04	Michael Pizzo, Ralf Handl, Martin Zurmuehl	Non-Material Changes
OASIS Specification	2014-02-24	Michael Pizzo, Ralf Handl, Martin Zurmuehl	Non-Material Changes
Errata 01	2014-07-24	Michael Pizzo, Ralf Handl, Martin Zurmuehl	Minor changes and improvements
Errata 02	2014-10-29	Michael Pizzo, Ralf Handl, Martin Zurmuehl	Repaired mechanical error in the editable source

OData Version 4.0 Part 3: Common Schema Definition Language (CSDL) Plus Errata 02

OASIS Standard incorporating Approved Errata 02

30 October 2014

Specification URIs

This version:

http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part3-csdl/odata-v4.0-errata02-os-part3-csdl-complete.doc (Authoritative)
http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part3-csdl/odata-v4.0-errata02-os-part3-csdl-complete.html
http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part3-csdl/odata-v4.0-errata02-os-part3-csdl-complete.pdf

Previous version:

http://docs.oasis-open.org/odata/odata/v4.0/errata01/os/complete/part3-csdl/odata-v4.0-errata01-os-part3-csdl-complete.doc (Authoritative)
http://docs.oasis-open.org/odata/odata/v4.0/errata01/os/complete/part3-csdl/odata-v4.0-errata01-os-part3-csdl-complete.html
http://docs.oasis-open.org/odata/odata/v4.0/errata01/os/complete/part3-csdl/odata-v4.0-errata01-os-part3-csdl-complete.pdf

Latest version:

http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part3-csdl.doc (Authoritative)
http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part3-csdl.html
http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part3-csdl.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:

Mike 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 02*. Edited by Michael Pizzo, Ralf Handl, Martin Zurmuehl, and Hubert Heijkers. 30 October 2014. OASIS Approved Errata. http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/odata-v4.0-errata02-os.html.
- *OData Version 4.0 Part 1: Protocol Plus Errata 02*. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. 30 October 2014. OASIS Standard incorporating Approved Errata 02.

http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part1-protocol/odata-v4.0-errata02-os-part1-protocol-complete.html.

- *OData Version 4.0 Part 2: URL Conventions Plus Errata 02*. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. 30 October 2014. OASIS Standard incorporating Approved Errata 02. http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part2-url-conventions/odata-v4.0-errata02-os-part2-url-conventions-complete.html.
- *OData Version 4.0 Part 3: Common Schema Definition Language (CSDL) Plus Errata 02* (this document). Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. 30 October 2014. OASIS Standard incorporating Approved Errata 02. http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part3-csdl/odata-v4.0-errata02-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/errata02/os/complete/abnf/.
- Vocabulary components: OData Core Vocabulary, OData Measures Vocabulary and OData Capabilities Vocabulary. http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/vocabularies/.
- XML schemas: OData EDMX XML Schema and OData EDM XML Schema. http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/schemas/.
- OData Metadata Service Entity Model: http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/models/.
- Change-marked (redlined) versions of OData Version 4.0 Part 1, Part 2, and Part 3. OASIS Standard incorporating Approved Errata 02. http://docs.oasis-open.org/odata/odata/v4.0/errata02/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:

OData services are described by an Entity Data Model (EDM). The Common Schema Definition Language (CSDL) defines an XML representation of the entity data model exposed by an OData service.

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-Part3]

OData Version 4.0 Part 3: Common Schema Definition Language (CSDL) Plus Errata 02. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. 30 October 2014. OASIS Standard incorporating Approved Errata 02. http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part3-csdl/odata-v4.0-errata02-os-part3-csdl-complete.html. Latest version: http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part3-csdl.html.

Notices

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

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

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

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

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 CSDL Namespaces 12
2.1 Namespace EDMX 12
2.2 Namespace EDM 12
2.3 XML Schema Definitions 12
2.4 XML Document Order 13
3 Entity Model Wrapper 14
3.1 Element `edmx:Edmx` 14
3.1.1 Attribute `Version` 14
3.2 Element `edmx:DataServices` 14
3.3 Element `edmx:Reference` 14
3.3.1 Attribute `Uri` 15
3.4 Element `edmx:Include` 15
3.4.1 Attribute `Namespace` 15
3.4.2 Attribute `Alias` 15
3.5 Element `edmx:IncludeAnnotations` 16
3.5.1 Attribute `TermNamespace` 16
3.5.2 Attribute `Qualifier` 16
3.5.3 Attribute `TargetNamespace` 17
4 Common Characteristics of Entity Models 18
4.1 Nominal Types 18
4.2 Structured Types 18
4.3 Structural Properties 18
4.4 Primitive Types 18
4.5 Built-In Abstract Types 20
4.6 Annotations 20
5 Schema 21
5.1 Element `edm:Schema` 21
5.1.1 Attribute `Namespace` 21
5.1.2 Attribute `Alias` 21
6 Structural Property 22
6.1 Element `edm:Property` 22
6.1.1 Attribute `Name` 22
6.1.2 Attribute `Type` 22
6.2 Property Facets 22
6.2.1 Attribute `Nullable` 22
6.2.2 Attribute `MaxLength` 23
6.2.3 Attribute `Precision` 23
6.2.4 Attribute `Scale` 23

6.2.5 Attribute Unicode 24
6.2.6 Attribute SRID 24
6.2.7 Attribute DefaultValue 24
7 Navigation Property 25
7.1 Element edm:NavigationProperty 25
7.1.1 Attribute Name 25
7.1.2 Attribute Type 25
7.1.3 Attribute Nullable 25
7.1.4 Attribute Partner 26
7.1.5 Attribute ContainsTarget 26
7.2 Element edm:ReferentialConstraint 27
7.2.1 Attribute Property 27
7.2.2 Attribute ReferencedProperty 27
7.3 Element edm:OnDelete 27
7.3.1 Attribute Action 27
8 Entity Type 29
8.1 Element edm:EntityType 29
8.1.1 Attribute Name 29
8.1.2 Attribute BaseType 29
8.1.3 Attribute Abstract 30
8.1.4 Attribute OpenType 30
8.1.5 Attribute HasStream 30
8.2 Element edm:Key 30
8.3 Element edm:PropertyRef 31
8.3.1 Attribute Name 31
8.3.2 Attribute Alias 32
9 Complex Type 33
9.1 Element edm:ComplexType 33
9.1.1 Attribute Name 33
9.1.2 Attribute BaseType 33
9.1.3 Attribute Abstract 33
9.1.4 Attribute OpenType 34
10 Enumeration Type 35
10.1 Element edm:EnumType 35
10.1.1 Attribute Name 35
10.1.2 Attribute UnderlyingType 35
10.1.3 Attribute IsFlags 35
10.2 Element edm:Member 35
10.2.1 Attribute Name 35
10.2.2 Attribute Value 36
11 Type Definition 37
11.1 Element edm:TypeDefinition 37
11.1.1 Attribute Name 37

11.1.2 Attribute UnderlyingType 37
11.1.3 Type Definition Facets 37
12 Action and Function 38
12.1 Element edm:Action 38
12.1.1 Attribute Name 38
12.1.1.1 Action Overload Rules 38
12.1.2 Attribute IsBound 38
12.1.3 Attribute EntitySetPath 38
12.2 Element edm:Function 39
12.2.1 Attribute Name 39
12.2.1.1 Function Overload Rules 39
12.2.2 Attribute IsBound 39
12.2.3 Attribute IsComposable 39
12.2.4 Attribute EntitySetPath 40
12.3 Element edm:ReturnType 40
12.3.1 Attribute Type 40
12.3.2 Attribute Nullable 40
12.4 Element edm:Parameter 40
12.4.1 Attribute Name 40
12.4.2 Attribute Type 41
12.4.3 Attribute Nullable 41
12.4.4 Parameter Facets 41
13 Entity Container 42
13.1 Element edm:EntityContainer 43
13.1.1 Attribute Name 43
13.1.2 Attribute Extends 43
13.2 Element edm:EntitySet 43
13.2.1 Attribute Name 43
13.2.2 Attribute EntityType 43
13.2.3 Attribute IncludeInServiceDocument 43
13.3 Element edm:Singleton 44
13.3.1 Attribute Name 44
13.3.2 Attribute Type 44
13.4 Element edm:NavigationPropertyBinding 44
13.4.1 Attribute Path 44
13.4.2 Attribute Target 44
13.5 Element edm:ActionImport 45
13.5.1 Attribute Name 45
13.5.2 Attribute Action 45
13.5.3 Attribute EntitySet 45
13.6 Element edm:FunctionImport 45
13.6.1 Attribute Name 45
13.6.2 Attribute Function 45

13.6.3 Attribute EntitySet 45
13.6.4 Attribute IncludeInServiceDocument 46
14 Vocabulary and Annotation 47
14.1 Element edm:Term 48
14.1.1 Attribute Name 48
14.1.2 Attribute Type 48
14.1.3 Attribute BaseTerm 48
14.1.4 Attribute DefaultValue 48
14.1.5 Attribute AppliesTo 48
14.1.6 Term Facets 48
14.2 Element edm:Annotations 49
14.2.1 Attribute Target 49
14.2.2 Attribute Qualifier 50
14.3 Element edm:Annotation 50
14.3.1 Attribute Term 51
14.3.2 Attribute Qualifier 51
14.4 Constant Expressions 51
14.4.1 Expression edm:Binary 51
14.4.2 Expression edm:Bool 51
14.4.3 Expression edm:Date 52
14.4.4 Expression edm:DateTimeOffset 52
14.4.5 Expression edm:Decimal 52
14.4.6 Expression edm:Duration 53
14.4.7 Expression edm:EnumMember 53
14.4.8 Expression edm:Float 53
14.4.9 Expression edm:Guid 53
14.4.10 Expression edm:Int 54
14.4.11 Expression edm:String 54
14.4.12 Expression edm:TimeOfDay 54
14.5 Dynamic Expressions 54
14.5.1 Comparison and Logical Operators 55
14.5.2 Expression edm:AnnotationPath 55
14.5.3 Expression edm:Apply 56
14.5.3.1 Attribute Function 56
14.5.3.1.1 Function odata.concat 56
14.5.3.1.2 Function odata.fillUriTemplate 56
14.5.3.1.3 Function odata.uriEncode 57
14.5.4 Expression edm:Cast 57
14.5.4.1 Attribute Type 57
14.5.5 Expression edm:Collection 57
14.5.6 Expression edm:If 58
14.5.7 Expression edm:IsOf 58
14.5.7.1 Attribute Type 58

14.5.8 Expression edm:LabeledElement 59
14.5.8.1 Attribute Name 59
14.5.9 Expression edm:LabeledElementReference 59
14.5.10 Expression edm:Null 59
14.5.11 Expression edm:NavigationPropertyPath 60
14.5.12 Expression edm:Path 60
14.5.13 Expression edm:PropertyPath 61
14.5.14 Expression edm:Record 62
14.5.14.1 Attribute Type 63
14.5.14.2 Element edm:PropertyValue 63
14.5.14.2.1 Attribute Property 63
14.5.15 Expression edm:UrlRef 63
15 Metadata Service Schema 64
15.1 Entity Model Wrapper 65
15.2 Schema 66
15.3 Types 67
15.4 Properties 68
15.5 Actions and Functions 71
15.6 Entity Container 72
15.7 Terms and Annotations 74
16 CSDL Examples 77
16.1 Products and Categories Example 77
16.2 Annotations for Products and Categories Example 79
17 Attribute Values 80
17.1 Namespace 80
17.2 SimpleIdentifier 80
17.3 QualifiedName 80
17.4 TypeName 80
17.5 TargetPath 80
17.6 Boolean 81
18 Conformance 82
Appendix A. Acknowledgments 83
Appendix B. Revision History 84

1 Introduction

OData services are described in terms of an Entity Data Model (EDM). The Common Schema Definition Language (CSDL) defines an XML representation of the entity data model exposed by an OData service. CSDL is articulated in the Extensible Markup Language (XML) 1.1 (Second Edition) **[XML-1.1]** with further building blocks from the W3C XML Schema Definition Language (XSD) 1.1 as described in **[XML-Schema-1]** and **[XML-Schema-2]**.

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

[EPSG] European Petroleum Survey Group (EPSG). http://www.epsg.org/.

[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-EDM] *OData EDM XML Schema*.
See link in “Additional artifacts” section on cover page.

[OData-EDMX] *OData EDMX XML Schema*.
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-Meta] *OData Metadata Service Schema.*
See link in “Additional artifacts” section on cover page.

[OData-Protocol] *OData Version 4.0 Part 1: Protocol*.
See link in “Additional artifacts” section on cover page.

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

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

[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.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, “URI Template”, RFC 6570, March 2012. http://tools.ietf.org/html/rfc6570.

[XML-1.1] Extensible Markup Language (XML) 1.1 (Second Edition), F. Yergeau, E. Maler, J. Cowan, T. Bray, C. M. Sperberg-McQueen, J. Paoli, Editors, W3C Recommendation, 16 August 2006,
http://www.w3.org/TR/2006/REC-xml11-20060816.
Latest version available at http://www.w3.org/TR/xml11/.

[XML-Base] XML Base (Second Edition) , J. Marsh, R. Tobin, Editors, W3C Recommendation, 28 January 2009,
http://www.w3.org/TR/2009/REC-xmlbase-20090128/.
Latest version available at http://www.w3.org/TR/xmlbase/.

[XML-Schema-1] W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures, D. Beech, M. Maloney, C. M. Sperberg-McQueen, H. S. Thompson, S. Gao, N. Mendelsohn, Editors, W3C Recommendation, 5 April 2012,

http://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/.
Latest version available at http://www.w3.org/TR/xmlschema11-1/.

[XML-Schema-2] W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes, D. Peterson, S. Gao, C. M. Sperberg-McQueen, H. S. Thompson, P. V. Biron, A. Malhotra, Editors, W3C Recommendation, 5 April 2012, http://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/.
Latest version available at http://www.w3.org/TR/xmlschema11-2/.

1.3 Typographical Conventions

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

```
Normative source code uses this paragraph style.
```

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

Example 1: text describing an example uses this paragraph style

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

All examples in this document are non-normative and informative only.

All other text is normative unless otherwise labeled.

2 CSDL Namespaces

In addition to the default XML namespace, the elements and attributes used to describe the entity model of an OData service are defined in one of the following namespaces. An XML document using these namespaces and having an `edmx:Edmx` root element will be called a CSDL document.

2.1 Namespace EDMX

Elements and attributes associated with the top-level wrapper that contains the CSDL used to define the entity model for an OData Service are qualified with the Entity Data Model for Data Services Packaging namespace:

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

Prior versions of OData used the following namespace for EDMX:

- EDMX version 1.0: `http://schemas.microsoft.com/ado/2007/06/edmx`

They are non-normative for this specification.

In this specification the namespace prefix `edmx` is used to represent the Entity Data Model for Data Services Packaging namespace, however the prefix name is not prescriptive.

2.2 Namespace EDM

Elements and attributes that define the entity model exposed by the OData Service are qualified with the Entity Data Model namespace:

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

Prior versions of CSDL used the following namespaces for EDM:

- CSDL version 1.0: `http://schemas.microsoft.com/ado/2006/04/edm`
- CSDL version 1.1: `http://schemas.microsoft.com/ado/2007/05/edm`
- CSDL version 1.2: `http://schemas.microsoft.com/ado/2008/01/edm`
- CSDL version 2.0: `http://schemas.microsoft.com/ado/2008/09/edm`
- CSDL version 3.0: `http://schemas.microsoft.com/ado/2009/11/edm`

They are non-normative for this specification.

In this specification the namespace prefix `edm` is used to represent the Entity Data Model namespace, however the prefix name is not prescriptive.

2.3 XML Schema Definitions

This specification contains normative XML schemas for the EDMX and EDM namespaces; see **[OData-EDMX]** and **[OData-EDM]**.

These XML schemas only define the shape of a well-formed CSDL document, but are not descriptive enough to define what a correct CSDL document MUST be in every imaginable use case. This specification document defines additional rules that correct CSDL documents MUST fulfill. In case of doubt on what makes a CSDL document correct the rules defined in this specification document take precedence.

2.4 XML Document Order

Client libraries MUST retain the document order of XML elements for CSDL documents because for some elements the order of child elements is significant. This includes, but is not limited to, members of enumeration types and items within a collection-valued annotation.

OData does not impose any ordering constraints on XML attributes within XML elements.

3 Entity Model Wrapper

An OData service exposes a single entity model. This model may be distributed over several schemas, and these schemas may be distributed over several physical locations. The entity model wrapper provides a single point of access to these parts by including them directly or referencing their physical locations.

A service is defined by a single CSDL document which can be accessed by sending a `GET` request to `<serviceRoot>/$metadata`. This document is called the metadata document. It may reference other CSDL documents.

The metadata document contains a single entity container that defines the resources exposed by this service. This entity container MAY extend an entity container defined in referenced documents.

The *model* of the service consists of all CSDL constructs used in its entity containers.

3.1 Element `edmx:Edmx`

A CSDL document MUST contain a root `edmx:Edmx` element. This element MUST contain a single direct child `edmx:DataServices` element. In addition to the data services element, the `Edmx` element contains zero or more `edmx:Reference` elements.

Example 2:

```
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx"
           Version="4.0">
  <edmx:DataServices>
    ...
  </edmx:DataServices>
</edmx:Edmx>
```

3.1.1 Attribute `Version`

The `edmx:Edmx` element MUST provide the value `4.0` for the `Version` attribute. It specifies the version of the EDMX wrapper defined by this version of the specification.

3.2 Element `edmx:DataServices`

The `edmx:DataServices` element MUST contain one or more `edm:Schema` elements which define the schemas exposed by the OData service.

3.3 Element `edmx:Reference`

The `edmx:Reference` element specifies external CSDL documents referenced by the referencing document. The child elements `edmx:Include` and `edmx:IncludeAnnotations` specify which parts of the referenced document are available for use in the referencing document. The `edmx:Reference` element MUST contain at least one `edmx:Include` or `edmx:IncludeAnnotations` child element.

The `edmx:Reference` element contains zero or more `edm:Annotation` elements.

The *scope* of a CSDL document is the document itself and all schemas included from directly referenced documents. All entity types, complex types and other named elements *in scope* (that is, defined in the document itself or a schema of a directly referenced document) can be accessed from a referencing document by their namespace-qualified names.

Referencing another document may alter the model defined by the referencing document. For instance, if a referenced document defines an entity type derived from an entity type in the referencing document, then an entity set of the service defined by the referencing document may return entities of the derived

type. This is identical to the behavior if the derived type had been defined directly in the referencing document.

Note: referencing documents is not recursive. Only named elements defined in directly referenced documents can be used within the schema. However, those elements may in turn include elements defined in schemas referenced by their defining schema.

Example 3: to reference entity models containing definitions of vocabulary terms

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx"
           Version="4.0">
  <edmx:Reference Uri="http://vocabs.odata.org/capabilities/v1">
    <edmx:Include Namespace="Org.OData.Capabilities.V1" />
  </edmx:Reference>
  <edmx:Reference Uri="http://vocabs.odata.org/display/v1">
    <edmx:Include Alias="UI" Namespace="org.example.Display" />
  </edmx:Reference>
  <edmx:DataServices>...</edmx:DataServices>
</edmx:Edmx>
```

3.3.1 Attribute `Uri`

The `edmx:Reference` element MUST specify a `Uri` attribute. The `Uri` attribute uniquely identifies a model, so two references MUST NOT specify the same URI. The value of the `Uri` attribute SHOULD be a URL that locates a CSDL document describing the referenced model. If the URI is not dereferencable it SHOULD identify a well-known schema. The value of the `Uri` attribute MAY be an absolute or relative URI; relative URIs are relative to the `xml:base` attribute, see **[XML-Base]**.

3.4 Element `edmx:Include`

The `edmx:Reference` element contains zero or more `edmx:Include` elements that specify the schemas to include from the target document.

3.4.1 Attribute `Namespace`

The `edmx:Include` element MUST provide a Namespace value for the `Namespace` attribute. The value MUST match the namespace of a schema defined in the referenced CSDL document. The same namespace MUST NOT be included more than once, even if it is declared in more than one referenced document.

3.4.2 Attribute `Alias`

An `edmx:Include` element MAY define a SimpleIdentifier value for the `Alias` attribute. The `Alias` attribute defines an alias for the specified `Namespace` that can be used in qualified names instead of the namespace. It only provides a more convenient notation. Every model element that can be used via an alias-qualified name can alternatively also be used via its full namespace-qualified name. An alias allows a short string to be substituted for a long namespace. For instance, an alias of `display` might be assigned to the namespace `org.example.vocabularies.display`. An alias-qualified name is resolved to a fully qualified name by examining aliases on `edmx:Include` and `edm:Schema` elements within the same document.

Aliases are document-global, so `edmx:Include` and `edm:Schema` elements within a document MUST NOT assign the same alias to different namespaces and MUST NOT specify an alias with the same name as an in-scope namespace.

The `Alias` attribute MUST NOT use the reserved values `Edm`, `odata`, `System`, or `Transient`.

An alias is only valid within the document in which it is declared; a referencing document has to define its own aliases with the `edmx:Include` element.

3.5 Element edmx:IncludeAnnotations

The edmx:Reference element contains zero or more edmx:IncludeAnnotations elements that specify the annotations to include from the target document. If no edmx:IncludeAnnotations element is specified, a client MAY ignore all annotations in the referenced document that are not explicitly used in an edm:Path expression of the referencing document.

Example 4: reference documents that contain annotations

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx"
           Version="4.0">
  <edmx:Reference Uri="http://odata.org/ann/b">
    <edmx:IncludeAnnotations TermNamespace="org.example.validation" />
    <edmx:IncludeAnnotations TermNamespace="org.example.display"
                             Qualifier="Tablet" />
    <edmx:IncludeAnnotations TermNamespace="org.example.hcm"
                             TargetNamespace="com.contoso.Sales" />
    <edmx:IncludeAnnotations TermNamespace="org.example.hcm"
                             Qualifier="Tablet"
                             TargetNamespace="com.contoso.Person" />
  </edmx:Reference>
  <edmx:DataServices>...</edmx:DataServices>
</edmx:Edmx>
```

The following annotations from `http://odata.org/ann/b` are included:

- *Annotations that use a term from the `org.example.validation` namespace, and*
- *Annotations that use a term from the `org.example.display` namespace and specify a `Tablet` qualifier and*
- *Annotations that apply a term from the `org.example.hcm` namespace to an element of the `com.contoso.Sales` namepace and*
- *Annotations that apply a term from the `org.example.hcm` namespace to an element of the `com.contoso.Person` namepace and specify a `Tablet` qualifier.*

3.5.1 Attribute TermNamespace

An edmx:IncludeAnnotations element MUST provide a Namespace value for the TermNamespace attribute.

The edmx:IncludeAnnotations element will import the set of annotations that apply terms defined in the schema identified by the TermNamespace value. The TermNamespace attribute also provides consumers insight about what namespaces are used in the annotations document. If there are no edmx:IncludeAnnotations elements that have a term namespace of interest to the consumer, the consumer can opt not to download the document.

3.5.2 Attribute Qualifier

An edmx:IncludeAnnotations element MAY specify a SimpleIdentifier for the Qualifier attribute. A qualifier is used to apply an annotation to a subset of consumers. For instance, a service author might want to supply a different set of annotations for various device form factors.

If Qualifier is specified, only those annotations applying terms from the specified TermNamespace with the specified Qualifier (applied to an element of the TargetNamespace, if present) SHOULD be imported. If Qualifier is not specified, all annotations within the referenced document from the specified TermNamespace (taking into account the TargetNamespace, if present) SHOULD be imported.

The `Qualifier` attribute also provides consumers insight about what qualifiers are used in the annotations document. If the consumer is not interested in that particular qualifier, the consumer can opt not to download the document.

3.5.3 Attribute `TargetNamespace`

An `edmx:IncludeAnnotations` element MAY specify a Namespace value for the `TargetNamespace` attribute.

If `TargetNamespace` is specified, only those annotations which apply a term from the specified `TermNamespace` to an element of the `TargetNamespace` (with the specified `Qualifier`, if present) SHOULD be imported. If `TargetNamespace` is not specified, all annotations within the referenced document from the specified `TermNamespace` (taking into account the `Qualifier`, if present) SHOULD be imported.

The `TargetNamespace` attribute also provides consumers insight about what namespaces are used in the annotations document. If there are no target elements that have a namespace of interest to the consumer, the consumer can opt not to download the document.

4 Common Characteristics of Entity Models

4.1 Nominal Types

A nominal type has a name that MUST be a SimpleIdentifier. Nominal types are referenced using their QualifiedName. The qualified type name MUST be unique within a model as it facilitates references to the element from other parts of the model.

When referring to nominal types, the reference MUST use one of the following:

- Namespace-qualified name
- Alias-qualified name

Example 5:

```
<Schema xmlns="http://docs.oasis-open.org/odata/ns/edm"
        Namespace="org.example"
        Alias="sales">
  <ComplexType Name="Address">...</ComplexType>
</Schema>
```

The two ways of referring to the nominal type `Address` are:

- *the fully qualified name `org.example.Address` can be used in any namespace*
- *an alias could be specified in any namespace and used in an alias-qualified name, e.g. `sales.Address`*

4.2 Structured Types

Structured types are composed of other model elements. Structured types are common in entity models as the means of representing entities and structured properties in an OData service. Entity types and complex types are both structured types.

4.3 Structural Properties

A structural property is a property (of a structural type) that has one of the following types:

- Primitive type
- Complex type
- Enumeration type
- A collection of one of the above

4.4 Primitive Types

Structured types are composed of other structured types and primitive types. OData defines the following primitive types:

Type	Meaning
`Edm.Binary`	Binary data
`Edm.Boolean`	Binary-valued logic
`Edm.Byte`	Unsigned 8-bit integer
`Edm.Date`	Date without a time-zone offset

Type	Meaning
Edm.DateTimeOffset	Date and time with a time-zone offset, no leap seconds
Edm.Decimal	Numeric values with fixed precision and scale
Edm.Double	IEEE 754 binary64 floating-point number (15-17 decimal digits)
Edm.Duration	Signed duration in days, hours, minutes, and (sub)seconds
Edm.Guid	16-byte (128-bit) unique identifier
Edm.Int16	Signed 16-bit integer
Edm.Int32	Signed 32-bit integer
Edm.Int64	Signed 64-bit integer
Edm.SByte	Signed 8-bit integer
Edm.Single	IEEE 754 binary32 floating-point number (6-9 decimal digits)
Edm.Stream	Binary data stream
Edm.String	Sequence of UTF-8 characters
Edm.TimeOfDay	Clock time 00:00-23:59:59.999999999999
Edm.Geography	Abstract base type for all Geography types
Edm.GeographyPoint	A point in a round-earth coordinate system
Edm.GeographyLineString	Line string in a round-earth coordinate system
Edm.GeographyPolygon	Polygon in a round-earth coordinate system
Edm.GeographyMultiPoint	Collection of points in a round-earth coordinate system
Edm.GeographyMultiLineString	Collection of line strings in a round-earth coordinate system
Edm.GeographyMultiPolygon	Collection of polygons in a round-earth coordinate system
Edm.GeographyCollection	Collection of arbitrary Geography values
Edm.Geometry	Abstract base type for all Geometry types
Edm.GeometryPoint	Point in a flat-earth coordinate system
Edm.GeometryLineString	Line string in a flat-earth coordinate system
Edm.GeometryPolygon	Polygon in a flat-earth coordinate system
Edm.GeometryMultiPoint	Collection of points in a flat-earth coordinate system
Edm.GeometryMultiLineString	Collection of line strings in a flat-earth coordinate system
Edm.GeometryMultiPolygon	Collection of polygons in a flat-earth coordinate system
Edm.GeometryCollection	Collection of arbitrary Geometry values

Edm.Date and Edm.DateTimeOffset follow **[XML-Schema-2]** and use the proleptic Gregorian calendar, allowing the year 0000 and negative years.

Edm.Stream is a primitive type that can be used as a property of an entity type or complex type, the underlying type for a type definition, or the binding parameter or return type of a function or action.

`Edm.Stream`, or a type definition whose underlying type is `Edm.Stream`, cannot be used in collections or for non-binding parameters to functions or actions.

Some of these types allow facet attributes, defined in section 6.2.

See rule `primitiveLiteral` in **[OData-ABNF]** for the representation of primitive type values in URLs, and **[OData-Atom]** and **[OData-JSON]** for the representation in requests and responses.

4.5 Built-In Abstract Types

The following built-in abstract types can be used within a model:

- `Edm.PrimitiveType`
- `Edm.ComplexType`
- `Edm.EntityType`

Conceptually, these are the abstract base types for primitive types (including type definitions and enumeration types), complex types, and entity types, respectively, and can be used anywhere a corresponding concrete type can be used, except:

- `Edm.EntityType`
  - cannot be used as the type of a singleton in an entity container because it doesn't define a structure, which defeats the purpose of a singleton.
  - cannot be used as the type of an entity set because all entities in an entity set must have the same key fields to uniquely identify them within the set.
  - cannot be the base type of an entity type or complex type.
- `Edm.ComplexType`
  - cannot be the base type of an entity type or complex type.
- `Edm.PrimitiveType`
  - cannot be used as the type of a key property of an entity type.
  - cannot be used as the underlying type of a type definition or enumeration type.
- `Collection(Edm.PrimitiveType)` and `Collection(Edm.ComplexType)`
  - cannot be used as the type of a property.
  - cannot be used as the return type of a function.

Vocabulary terms can, in addition, use

- `Edm.AnnotationPath`
- `Edm.PropertyPath`
- `Edm.NavigationPropertyPath`

as the type of a primitive term, or the type of a property of a complex type that is exclusively used as the type of a term.

4.6 Annotations

Many parts of the model can be annotated with additional information using the `edm:Annotation` element.

A model element MUST NOT specify more than one annotation for a given combination of `Term` and `Qualifier` attributes.

Vocabulary annotations can be specified as a child of the model element being annotated or as a child of an `edm:Annotations` element that targets the model element.

Refer to Vocabulary Annotations for details on which model elements support vocabulary annotations.

5 Schema

One or more schemas describe the entity model exposed by an OData service. The schema acts as a namespace for elements of the entity model such as entity types, complex types, enumerations and terms.

5.1 Element edm:Schema

The edm:Schema element contains one or more of the following elements:

- edm:Action
- edm:Annotations
- edm:Annotation
- edm:ComplexType
- edm:EntityContainer
- edm:EntityType
- edm:EnumType
- edm:Function
- edm:Term
- edm:TypeDefinition

Values of the Name attribute MUST be unique across all direct child elements of a schema, with the sole exception of overloads for an action and overloads for a function. The names are local to the schema; they need not be unique within a document.

5.1.1 Attribute Namespace

A schema is identified by a namespace. All edm:Schema elements MUST have a namespace defined through a Namespace attribute which MUST be unique within the document, and SHOULD be globally unique. A schema cannot span more than one document.

The schema's namespace is combined with the name of elements in the entity model to create unique qualified names, so identifiers that are used to name types MUST be unique within a namespace to prevent ambiguity. See Nominal Types for more detail.

The Namespace attribute MUST NOT use the reserved values Edm, odata, System, or Transient.

5.1.2 Attribute Alias

A schema MAY define an alias by providing a SimpleIdentifier value for the Alias attribute. An alias allows nominal types to be qualified with a short string rather than a long namespace.

Aliases are document-global, so all edmx:Include and edm:Schema elements within a document MUST specify different values for the Alias attribute. Aliases defined by an edm:Schema element can be used throughout the containing document and are not restricted to the schema that defines them.

The Alias attribute MUST NOT use the reserved values Edm, odata, System, or Transient.

6 Structural Property

Structured Types are composed of zero or more structural properties (represented as `edm:Property` elements) and navigation properties (represented as `edm:NavigationProperty` elements).

Example 6: complex type with two properties

```
<ComplexType Name="Measurement">
  <Property Name="Dimension" Type="Edm.String" Nullable="false" MaxLength="50"
            DefaultValue="Unspecified" />
  <Property Name="Length" Type="Edm.Decimal" Nullable="false" Precision="18"
            Scale="2" />
</ComplexType>
```

Open entity types and open complex types allow properties to be added dynamically to instances of the open type.

6.1 Element `edm:Property`

The `edm:Property` element defines a structural property.

Example 7: property that can have zero or more strings as its value

```
<Property Name="Units" Type="Collection(Edm.String)" />
```

A property MUST specify a unique name as well as a type and zero or more facets. Facets are attributes that modify or constrain the acceptable values for a property value.

6.1.1 Attribute `Name`

The `edm:Property` element MUST include a `Name` attribute whose value is a SimpleIdentifier used when referencing, serializing or deserializing the property.

The name of the structural property MUST be unique within the set of structural and navigation properties of the containing structured type and any of its base types.

6.1.2 Attribute `Type`

The `edm:Property` element MUST include a `Type` attribute. The value of the `Type` attribute MUST be the QualifiedName of a primitive type, complex type, or enumeration type in scope, or a collection of one of these types.

6.2 Property Facets

Property facets allow a model to provide additional constraints or data about the value of structural properties. Facets are expressed as attributes on the property element.

Facets apply to the type referenced in the element where the facet attribute is declared. If the type is a collection, the facets apply to the type of elements in the collection.

Example 8: `Precision` facet applied to the `DateTimeOffset` type

```
<Property Name="SuggestedTimes" Type="Collection(Edm.DateTimeOffset)"
          Precision="6" />
```

6.2.1 Attribute `Nullable`

The `edm:Property` element MAY contain the `Nullable` attribute whose Boolean value specifies whether a value is required for the property.

If no value is specified for a property whose `Type` attribute does not specify a collection, the `Nullable` attribute defaults to `true`.

If the `edm:Property` element contains a `Type` attribute that specifies a collection, the property MUST always exist, but the collection MAY be empty. In this case, the `Nullable` attribute applies to members of the collection and specifies whether the collection can contain null values.

6.2.2 Attribute `MaxLength`

A binary, stream or string property MAY define a positive integer value for the `MaxLength` facet attribute. The value of this attribute specifies the maximum length of the value of the property on a type instance. Instead of an integer value the constant `max` MAY be specified as a shorthand for the maximum length supported for the type by the service.

If no value is specified, the property has unspecified length.

6.2.3 Attribute `Precision`

A datetime-with-offset, decimal, duration, or time-of-day property MAY define a value for the `Precision` attribute.

For a decimal property the value of this attribute specifies the maximum number of digits allowed in the property's value; it MUST be a positive integer. If no value is specified, the decimal property has unspecified precision.

For a temporal property the value of this attribute specifies the number of decimal places allowed in the seconds portion of the property's value; it MUST be a non-negative integer between zero and twelve. If no value is specified, the temporal property has a precision of zero.

Note: service designers SHOULD be aware that some clients are unable to support a precision greater than 29 for decimal properties and 7 for temporal properties. Client developers MUST be aware of the potential for data loss when round-tripping values of greater precision. Updating via `PATCH` and exclusively specifying modified properties will reduce the risk for unintended data loss.

6.2.4 Attribute `Scale`

A decimal property MAY define a non-negative integer value or `variable` for the `Scale` attribute.

This attribute specifies the maximum number of digits allowed to the right of the decimal point.

The value `variable` means that the number of digits to the right of the decimal point may vary from zero to the value of the `Precision` attribute minus one. At least one digit is required to the left of the decimal point.

An integer value means that the number of digits to the right of the decimal point may vary from zero to the value of the Scale attribute, and the number of digits to the left of the decimal point may vary from one to the value of the Precision attribute minus the value of the Scale attribute.

The value of the `Scale` attribute MUST be less than or equal to the value of the `Precision` attribute. If no value is specified, the Scale facet defaults to zero.

Example 9: `Precision` and `Scale` facets applied to the `Decimal` type.
Allowed values: 1.23, 0.23, 3.14 and 0.7, not allowed values: 123, 12.3.

```
<Property Name="Amount" Type="Edm.Decimal" Precision="3" Scale="2" />
```

Example 10: `Precision` and `Scale` facets applied incorrectly to the `Decimal` type.
This is not allowed because .23 is not allowed anymore and 0.23 needs a Precision of 3.

```
<Property Name="Amount" Type="Edm.Decimal" Precision="2" Scale="2" />
```

Example 11: `Precision` and a variable `Scale` applied to the `Decimal` type.
Allowed values: 1.23, 0.23, 0.7, 123 and 12.3, not allowed would be: 12.34, 1234 and 123.4 due to the limited precision.


```
<Property Name="Amount" Type="Edm.Decimal" Precision="3" Scale="variable" />
```

6.2.5 Attribute Unicode

A string property MAY define a Boolean value for the Unicode attribute.

A true value assigned to this attribute indicates that the value of the property is encoded with Unicode. A false value assigned to this attribute indicates that the value of the property is encoded with ASCII.

If no value is specified, the Unicode facet defaults to true.

6.2.6 Attribute SRID

A geometry or geography property MAY define a value for the SRID attribute. The value of this attribute identifies which spatial reference system is applied to values of the property on type instances.

The value of the SRID attribute MUST be a non-negative integer or the special value variable. If no value is specified, the attribute defaults to 0 for Geometry types or 4326 for Geography types.

The valid values of the SRID attribute and their meanings are as defined by the European Petroleum Survey Group **[EPSG]**.

6.2.7 Attribute DefaultValue

A primitive or enumeration property MAY define a value for the DefaultValue attribute. The value of this attribute determines the value of the property if the property is not explicitly represented in an annotation or the body of a POST or PUT request.

Default values of type Edm.String MUST be represented according to the XML escaping rules for character data in attribute values. Values of other primitive types MUST be represented according to the appropriate alternative in the primitiveValue rule defined in **[OData-ABNF]**, i.e. Edm.Binary as binaryValue, Edm.Boolean as booleanValue etc.

If no value is specified, the client SHOULD NOT assume a default value.

7 Navigation Property

7.1 Element edm:NavigationProperty

A navigation property allows navigation to related entities.

Example 12: the Product entity type has a navigation property to a Category, which has a navigation link back to one or more products

```
<EntityType Name="Product">
  ...
  <NavigationProperty Name="Category" Type="Self.Category" Nullable="false"
                      Partner="Products" />
  <NavigationProperty Name="Supplier" Type="Self.Supplier" />
</EntityType>

<EntityType Name="Category">
  ...
  <NavigationProperty Name="Products" Type="Collection(Self.Product)"
                      Partner="Category" />
</EntityType>
```

7.1.1 Attribute Name

The `edm:NavigationProperty` element MUST include a `Name` attribute whose value is a SimpleIdentifier that is used when navigating from the structured type that declares the navigation property to the related entity type.

The name of the navigation property MUST be unique within the set of structural and navigation properties of the containing structured type and any of its base types.

7.1.2 Attribute Type

The `edm:NavigationProperty` element MUST include a `Type` attribute. The value of the type attribute MUST resolve to an entity type or a collection of an entity type declared in the same document or a document referenced with an `edmx:Reference` element, or the abstract type `Edm.EntityType`.

If the value is an entity type name, there can be at most one related entity. If it is a collection, an arbitrary number of entities can be related.

The related entities MUST be of the specified entity type or one of its subtypes.

7.1.3 Attribute Nullable

The `edm:NavigationProperty` element MAY contain the `Nullable` attribute whose Boolean value specifies whether a navigation target is required for the navigation property.

If no value is specified for a navigation property whose `Type` attribute does not specify a collection, the `Nullable` attribute defaults to `true`. The value `true` (or the absence of the `Nullable` attribute) indicates that no navigation target is required. The value `false` indicates that a navigation target is required for the navigation property on instances of the containing type.

A navigation property whose `Type` attribute specifies a collection MUST NOT specify a value for the `Nullable` attribute as the collection always exists, it may just be empty.

7.1.4 Attribute `Partner`

A navigation property of an entity type MAY specify a navigation property path value for the `Partner` attribute.

This attribute MUST NOT be specified for navigation properties of complex types.

If specified, the value of this attribute MUST be a path from the entity type specified in the `Type` attribute to a navigation property defined on that type or a derived type. The path may traverse complex types, including derived complex types, but MUST NOT traverse any navigation properties. The type of the partner navigation property MUST be the containing entity type of the current navigation property or one of its parent entity types.

If the `Partner` attribute identifies a single-valued navigation property, the partner navigation property MUST lead back to the source entity from all related entities. If the `Partner` attribute identifies a multi-valued navigation property, the source entity MUST be part of that collection.

If no partner navigation property is specified, no assumptions can be made as to whether one of the navigation properties on the target type will lead back to the source entity.

If a partner navigation property is specified, this partner navigation property MUST either specify the current navigation property as its partner to define a bi-directional relationship or it MUST NOT specify a partner attribute. The latter can occur if the partner navigation property is defined on a complex type or the current navigation property is defined on a type derived from the type of the partner navigation property.

7.1.5 Attribute `ContainsTarget`

A navigation property MAY assign a Boolean value to the `ContainsTarget` attribute. If no value is assigned to the `ContainsTarget` attribute, the attribute defaults to `false`. If the value of the `ContainsTarget` attribute is `true`, the navigation property is called a *containment navigation property*.

Containment navigation properties define an implicit entity set for each instance of its declaring entity type. This implicit entity set is identified by the read URL of the navigation property for that entity.

Entities of the entity type that declares the navigation property, either directly or indirectly via a property of complex type, contain the entities referenced by the containment navigation property. The canonical URL for contained entities is the canonical URL of the containing entity, followed by the path segment of the navigation property and the key of the contained entity, see **[OData-URL]**.

As items in a collection of complex types do not have a canonical URL, complex types declaring a containment navigation property, either directly or indirectly via a property of complex type, MUST NOT be used as the type of a collection-valued property.

An entity cannot be referenced by more than one containment relationship, and cannot both belong to an entity set declared within the entity container and be referenced by a containment relationship.

Containment navigation properties MUST NOT be specified as the last path segment in the `Path` attribute of a navigation property binding. When a containment navigation property navigates between entity types in the same inheritance hierarchy, the containment is called *recursive*.

Containment navigation properties MAY specify a `Partner` attribute. If the containment is recursive, the partner navigation property MUST be nullable and specify a single entity type. If the containment is not recursive, the partner navigation property MUST NOT be nullable.

An entity type hierarchy MUST NOT contain more than one navigation property with a `Partner` attribute referencing a containment relationship.

Note: without a partner attribute, there is no reliable way for a client to determine which entity contains a given contained entity. This may lead to problems for clients if the contained entity can also be reached via a non-containment navigation path.

7.2 Element edm:ReferentialConstraint

A navigation property whose Type attribute specifies a single entity type MAY define one or more referential constraints. A referential constraint asserts that the *dependent property* (the property defined on the *dependent entity* containing the navigation property) MUST have the same value as the *principal property* (the referenced property defined on the *principal entity* that is the target of the navigation).

The type of the dependent property MUST match the type of the principal property. If the navigation property on which the referential constraint is defined or the principal property is nullable, then the dependent property MUST be nullable. If both the navigation property and the principal property are not nullable, then the dependent property MUST be marked with the Nullable="false" attribute value.

Example 13: the category must exist for a product in that category to exist, and the CategoryID of the product is identical to the ID of the category

```
<EntityType Name="Product">
  ...
  <Property Name="CategoryID" Type="Edm.String" Nullable="false"/>
  <NavigationProperty Name="Category" Type="Self.Category" Nullable="false">
    <ReferentialConstraint Property="CategoryID" ReferencedProperty="ID" />
  </NavigationProperty>
</EntityType>
```

7.2.1 Attribute Property

A referential constraint MUST specify a value for the Property attribute. The Property attribute specifies the property that takes part in the referential constraint on the dependent entity type. Its value MUST be a path expression resolving to a primitive property of the dependent entity type itself or to a primitive property of a complex property (recursively) of the dependent entity type. The names of the properties in the path are joined together by forward slashes.

7.2.2 Attribute ReferencedProperty

A referential constraint MUST specify a value for the ReferencedProperty attribute. The ReferencedProperty attribute specifies the corresponding property of the principal entity type. Its value MUST be a path expression resolving to a primitive property of the principal entity type itself or to a primitive property of a complex property (recursively) of the principal entity type that MUST have the same data type as the property of the dependent entity type.

7.3 Element edm:OnDelete

A navigation property MAY define one edm:OnDelete element. It describes the action the service will take on related entities when the entity on which the navigation property is defined is deleted.

Example 14: deletion of a category implies deletion of the related products in that category

```
<EntityType Name="Category">
  ...
  <NavigationProperty Name="Products" Type="Collection(Self.Product)">
    <OnDelete Action="Cascade" />
  </NavigationProperty>
</EntityType>
```

7.3.1 Attribute Action

The edm:OnDelete element MUST include the Action attribute with one of the following values:

- Cascade, meaning the related entities will be deleted if the source entity is deleted,
- None, meaning a DELETE request on a source entity with related entities will fail,

- `SetNull`, meaning all properties of related entities that are tied to properties of the source entity via a referential constraint and that do not participate in other referential constraints will be set to null,
- `SetDefault`, meaning all properties of related entities that are tied to properties of the source entity via a referential constraint and that do not participate in other referential constraints will be set to their default value.

If no `edm:OnDelete` element is present, the action taken by the service is not predictable by the client and could vary per entity.

8 Entity Type

Entity types are nominal structured types with a key that consists of one or more references to structural properties. An entity type is the template for an entity: any uniquely identifiable record such as a customer or order.

An `edm.Key` child element MAY be specified if the entity type does not specify a base type that already has a key declared. The key consists of one or more references to structural properties of the entity type.

An entity type can define two types of properties. A structural property is a named reference to a primitive, complex, or enumeration type, or a collection of primitive, complex, or enumeration types. A navigation property is a named reference to another entity type or collection of entity types.

All properties MUST have a unique name within an entity type. Properties MUST NOT have the same name as the declaring entity type. They MAY have the same name as one of the direct or indirect base types or derived types.

An open entity type allows properties to be dynamically added to instances of the type.

Example 15: a simple entity type

```
<EntityType Name="Employee">
  <Key>
    <PropertyRef Name="ID" />
  </Key>
  <Property Name="ID" Type="Edm.String" Nullable="false" />
  <Property Name="FirstName" Type="Edm.String" Nullable="false" />
  <Property Name="LastName" Type="Edm.String" Nullable="false" />
  <NavigationProperty Name="Manager" Type="Model.Manager" />
</EntityType>
```

Example 16: a derived entity type based on the previous example

```
<EntityType Name="Manager" BaseType="Model.Employee">
  <Property Name="AnnualBudget" Type="Edm.Decimal" />
  <NavigationProperty Name="Employees" Type="Collection(Model.Employee)" />
</EntityType>
```

Note: the derived type has the same name as one of the properties of its base type.

8.1 Element `edm:EntityType`

The `edm:EntityType` element represents an entity type in the entity model. It contains zero or more `edm:Property` and `edm:NavigationProperty` elements describing the properties if the entity type.

It MAY contain one `edm:Key` element.

8.1.1 Attribute `Name`

The `edm:EntityType` element MUST include a `Name` attribute whose value is a SimpleIdentifier. The name MUST be unique within its namespace.

8.1.2 Attribute `BaseType`

An entity type can inherit from another entity type by specifying the QualifiedName of the base entity type as the value for the `BaseType` attribute.

An entity type inherits the key as well as structural and navigation properties declared on the entity type's base type.

An entity type MUST NOT introduce an inheritance cycle via the base type attribute.

8.1.3 Attribute Abstract

An entity type MAY indicate that it cannot be instantiated by providing a Boolean value of `true` to the `Abstract` attribute. If not specified, the `Abstract` attribute defaults to `false`.

If `Abstract` is `false`, the entity type MUST define a key or derive from a base type with a defined key.

An abstract entity type MUST NOT inherit from a non-abstract entity type.

8.1.4 Attribute OpenType

An entity type MAY indicate that it is open by providing a value of `true` for the `OpenType` attribute. An open type allows clients to add properties dynamically to instances of the type by specifying uniquely named values in the payload used to insert or update an instance of the type.

If not specified, the value of the `OpenType` attribute defaults to `false`.

An entity type derived from an open entity type MUST NOT provide a value of `false` for the `OpenType` attribute.

Note: structural and navigation properties MAY be returned by the service on instances of any structured type, whether or not the type is marked as open. Clients MUST always be prepared to deal with additional properties on instances of any structured type, see **[OData-Protocol]**.

8.1.5 Attribute HasStream

An entity type that does not specify a `BaseType` attribute MAY specify a Boolean value for the `HasStream` attribute.

A value of `true` specifies that the entity type is a media entity. *Media entities* are entities that represent a media stream, such as a photo. For more information on media entities see **[OData-Protocol]**.

If no value is provided for the `HasStream` attribute, and no `BaseType` attribute is specified, the value of the `HasStream` attribute is set to `false`.

The value of the the `HasStream` attribute is inherited by all derived types.

Entity types that specify `HasStream="true"` MAY specify a list of acceptable media types using an annotation with term `Core.AcceptableMediaTypes`, see **[OData-VocCore]**.

8.2 Element edm:Key

An entity is uniquely identified within an entity set by its key. An entity type that is not abstract MUST either contain exactly one `edm:Key` element or inherit its key from its base type. An abstract entity type MAY define a key if it doesn't inherit one.

An entity type's key refers to the set of properties that uniquely identify an instance of the entity type within an entity set.

The `edm:Key` element MUST contain at least one `edm:PropertyRef` element. An `edm:PropertyRef` element references an `edm:Property`. The properties that compose the key MUST be non-nullable and typed with an enumeration type, one of the following primitive types, or a type definition based on one of these primitive types:

- `Edm.Boolean`
- `Edm.Byte`
- `Edm.Date`
- `Edm.DateTimeOffset`
- `Edm.Decimal`
- `Edm.Duration`
- `Edm.Guid`

- Edm.Int16
- Edm.Int32
- Edm.Int64
- Edm.SByte
- Edm.String
- Edm.TimeOfDay

The properties that make up a primary key MAY be language-dependent, but their values MUST be unique across all languages and the entity ids (defined in **[OData-Protocol]**) MUST be language independent.

Example 17: entity type with a simple key

```
<EntityType Name="Category">
  <Key>
    <PropertyRef Name="ID" />
  </Key>
  <Property Name="ID" Type="Edm.Int32" Nullable="false" />
  <Property Name="Name" Type="Edm.String" />
</EntityType>
```

Example 18: entity type with a simple key referencing a property of a complex type

```
<EntityType Name="Category">
  <Key>
    <PropertyRef Name="Info/ID" Alias="EntityInfoID" />
  </Key>
  <Property Name="Info" Type="Sales.EntityInfo" Nullable="false" />
  <Property Name="Name" Type="Edm.String" />
</EntityType>

<ComplexType Name="EntityInfo">
  <Property Name="ID" Type="Edm.Int32" Nullable="false" />
  <Property Name="Created" Type="Edm.DateTimeOffset" />
</ComplexType>
```

Example 19: entity type with a composite key

```
<EntityType Name="OrderLine">
  <Key>
    <PropertyRef Name="OrderID" />
    <PropertyRef Name="LineNumber" />
  </Key>
  <Property Name="OrderID" Type="Edm.Int32" Nullable="false" />
  <Property Name="LineNumber" Type="Edm.Int32" Nullable="false" />
</EntityType>
```

8.3 Element edm:PropertyRef

The edm:PropertyRef element provides an edm:Key with a reference to a property.

8.3.1 Attribute Name

The edm:PropertyRef element MUST specify a value for the Name attribute which MUST be a path expression resolving to a primitive property of the entity type itself or to a primitive property of a complex property (recursively) of the entity type. The names of the properties in the path are joined together by forward slashes.

8.3.2 Attribute `Alias`

If the property identified by the `Name` attribute is a member of a complex type, the `edm:PropertyRef` element MUST specify the `Alias` attribute.

The value of the `Alias` attribute MUST be a SimpleIdentifier and MUST be unique within the set of aliases, structural and navigation properties of the containing entity type and any of its base types.

The `Alias` attribute MUST NOT be defined if the key property is not a member of a complex type.

For keys that are members of complex types, the alias MUST be used in the key predicate of URLs instead of the value assigned to the `Name` attribute. The alias MUST NOT be used in the query part.

Example 20 (based on example 18): requests to an entity set `Categories` of type `Category` must use the alias

```
http://host/service/Categories(EntityInfoID=1)
```

Example 21 (based on example 18): in a query part the value assigned to the name attribute must be used

```
http://example.org/OData.svc/Categories?$filter=Info/ID le 100
```

9 Complex Type

Complex types are keyless nominal structured types. The lack of a key means that complex types cannot be referenced, created, updated or deleted independently of an entity type. Complex types allow entity models to group properties into common structures.

A complex type can define two types of properties. A structural property is a named reference to a primitive, complex, or enumeration type, or a collection of primitive, complex, or enumeration types. A navigation property is a named reference to an entity type or a collection of entity types.

All properties MUST have a unique name within a complex type. Properties MUST NOT have the same name as the declaring complex type. They MAY have the same name as one of the direct or indirect base types or derived types.

An open complex type allows properties to be dynamically added to instances of the type.

Example 22: a complex type used by two entity types

```
<ComplexType Name="Dimensions">
  <Property Name="Height" Nullable="false" Type="Edm.Decimal" />
  <Property Name="Weight" Nullable="false" Type="Edm.Decimal" />
  <Property Name="Length" Nullable="false" Type="Edm.Decimal" />
</ComplexType>

<EntityType Name="Product">
  ...
  <Property Name="ProductDimensions" Type="Self.Dimensions" />
  <Property Name="ShippingDimensions" Type="Self.Dimensions" />
</EntityType>

<EntityType Name="ShipmentBox">
  ...
  <Property Name="Dimensions" Type="Self.Dimensions" />
</EntityType>
```

9.1 Element `edm:ComplexType`

The `edm:ComplexType` element represents a complex type in an entity model. It contains zero or more `edm:Property` and `edm:NavigationProperty` elements describing properties of the complex type.

9.1.1 Attribute `Name`

The `edm:ComplexType` element MUST include a `Name` attribute whose value is a SimpleIdentifier. The value identifies the complex type and MUST be unique within its namespace.

9.1.2 Attribute `BaseType`

A complex type can inherit from another complex type by specifying the QualifiedName of the base complex type as the value for the `BaseType` attribute.

A complex type inherits the properties declared on the complex type's base type.

A complex type MUST NOT introduce an inheritance cycle via the base type attribute.

9.1.3 Attribute `Abstract`

A complex type MAY indicate that it cannot be instantiated by providing a Boolean value of `true` to the `Abstract` attribute.

If not specified, the `Abstract` attribute defaults to `false`.

9.1.4 Attribute `OpenType`

A complex type MAY indicate that it is open by providing a value of `true` for the `OpenType` attribute. An open type allows clients to add properties dynamically to instances of the type by specifying uniquely named values in the payload used to insert or update an instance of the type.

If not specified, the `OpenType` attribute defaults to `false`.

A complex type derived from an open complex type MUST NOT provide a value of `false` for the `OpenType` attribute.

Note: structural and navigation properties MAY be returned by the service on instances of any structured type, whether or not the type is marked as open. Clients MUST always be prepared to deal with additional properties on instances of any structured type, see **[OData-Protocol]**.

10 Enumeration Type

Enumeration types are nominal types that represent a series of related values. Enumeration types expose these related values as members of the enumeration.

The `IsFlags` attribute indicates that more than one member may be selected at a time.

Example 23: a simple flags-enabled enumeration

```
<EnumType Name="FileAccess" UnderlyingType="Edm.Int32" IsFlags="true">
  <Member Name="Read"   Value="1" />
  <Member Name="Write"  Value="2" />
  <Member Name="Create" Value="4" />
  <Member Name="Delete" Value="8" />
</EnumType>
```

10.1 Element `edm:EnumType`

The `edm:EnumType` element represents an enumeration type in an entity model.

The enumeration type element contains one or more child `edm:Member` elements defining the members of the enumeration type.

10.1.1 Attribute `Name`

The `edm:EnumType` element MUST include a `Name` attribute whose value is a SimpleIdentifier. The value identifies the enumeration type and MUST be unique within its namespace.

10.1.2 Attribute `UnderlyingType`

An enumeration type MAY include an `UnderlyingType` attribute to specify an underlying type whose value MUST be one of `Edm.Byte`, `Edm.SByte`, `Edm.Int16`, `Edm.Int32`, or `Edm.Int64`. If the `UnderlyingType` attribute is not specified, `Edm.Int32` is used as the underlying type.

10.1.3 Attribute `IsFlags`

An enumeration type MAY specify a Boolean value for the `IsFlags` attribute. A value of `true` indicates that the enumeration type allows multiple members to be selected simultaneously.

If no value is specified for this attribute, its value defaults to `false`.

10.2 Element `edm:Member`

The `edm:Member` element defines the discrete options for the enumeration type .

Example 24: an enumeration type with three discrete members

```
<EnumType Name="ShippingMethod">
  <Member Name="FirstClass" />
  <Member Name="TwoDay" />
  <Member Name="Overnight" />
</EnumType>
```

10.2.1 Attribute `Name`

Each `edm:Member` element MUST include a `Name` attribute whose value is a SimpleIdentifier. The enumeration type MUST NOT declare two members with the same name.

10.2.2 Attribute `Value`

The value of an enumeration member allows instances to be sorted by a property that has an enumeration member for its value.

If the `IsFlags` attribute has a value of `false`, either all members MUST specify an integer value for the `Value` attribute, or all members MUST NOT specify a value for the `Value` attribute. If no values are specified, the members are assigned consecutive integer values in the order of their appearance, starting with zero for the first member. Client libraries MUST preserve elements in document order.

If the `IsFlags` attribute has a value of `true`, a non-negative integer value MUST be specified for the `Value` attribute. A combined value is equivalent to the bitwise OR of the discrete values.

The value MUST be a valid value for the `UnderlyingType` of the enumeration type.

Example 25: `FirstClass` has a value of `0`, `TwoDay` a value of 1, and `Overnight` a value of 2.

```
<EnumType Name="ShippingMethod">
  <Member Name="FirstClass" />
  <Member Name="TwoDay" />
  <Member Name="Overnight" />
</EnumType>
```

Example 26: pattern values can be combined, and some combined values have explicit names

```
<EnumType Name="Pattern" UnderlyingType="Edm.Int32" IsFlags="true">
  <Member Name="Plain"             Value="0" />
  <Member Name="Red"               Value="1" />
  <Member Name="Blue"              Value="2" />
  <Member Name="Yellow"            Value="4" />
  <Member Name="Solid"             Value="8" />
  <Member Name="Striped"           Value="16" />
  <Member Name="SolidRed"          Value="9" />
  <Member Name="SolidBlue"         Value="10" />
  <Member Name="SolidYellow"       Value="12" />
  <Member Name="RedBlueStriped"    Value="19" />
  <Member Name="RedYellowStriped"  Value="21" />
  <Member Name="BlueYellowStriped" Value="22" />
</EnumType>
```

11 Type Definition

11.1 Element edm:TypeDefinition

A type definition defines a specialization of one of the primitive types.

Type definitions can be used wherever a primitive type is used (other than as the underlying type in a new type definition), and are type-comparable with their underlying types and any type definitions defined using the same underlying type.

11.1.1 Attribute Name

The edm:TypeDefinition element MUST include a Name attribute whose value is a SimpleIdentifier. The name identifies the type definition and MUST be unique within its namespace.

11.1.2 Attribute UnderlyingType

The edm:TypeDefinition element MUST provide the QualifiedName of a primitive type as the value of the UnderlyingType attribute. This type MUST NOT be another type definition.

11.1.3 Type Definition Facets

The edm:TypeDefinition element MAY specify facets applicable to the underlying type: MaxLength, Unicode, Precision, Scale, or SRID.

Additional facets appropriate for the underlying type MAY be specified when the type definition is used but the facets specified in the type definition MUST NOT be re-specified.

Annotations MAY be applied to a type definition, and are considered applied wherever the type definition is used. The use of a type definition MUST NOT specify an annotation specified in the type definition.

Where type definitions are used, the type definition is returned in place of the primitive type wherever the type is specified in a response.

Example 27:

```
<TypeDefinition Name="Length" UnderlyingType="Edm.Int32">
  <Annotation Term="Org.OData.Measurements.V1.Unit"
              String="Centimeters" />
</TypeDefinition>

<TypeDefinition Name="Weight" UnderlyingType="Edm.Int32">
  <Annotation Term="Org.OData.Measurements.V1.Unit"
              String="Kilograms" />
</TypeDefinition>

<ComplexType Name="Size">
  <Property Name="Height" Type="Self.Length" />
  <Property Name="Weight" Type="Self.Weight" />
</ComplexType>
```

12 Action and Function

12.1 Element `edm:Action`

The `edm:Action` element represents an action in an entity model.

Actions MAY have observable side effects and MAY return a single instance or a collection of instances of any type. Actions cannot be composed with additional path segments.

The action MAY specify a return type using the `edm:ReturnType` element. The return type must be a primitive, entity or complex type, or a collection of primitive, entity or complex types.

The action may also define zero or more `edm:Parameter` elements to be used during the execution of the action.

12.1.1 Attribute `Name`

The `edm:Action` element MUST include a `Name` attribute whose value is a SimpleIdentifier.

12.1.1.1 Action Overload Rules

Bound actions support overloading (multiple actions having the same name within the same namespace) by binding parameter type. The combination of action name and the binding parameter type MUST be unique within a namespace.

Unbound actions do not support overloads. The names of all unbound actions MUST be unique within a namespace.

An unbound action MAY have the same name as a bound action.

12.1.2 Attribute `IsBound`

An action element MAY specify a Boolean value for the `IsBound` attribute.

Actions whose `IsBound` attribute is `false` or not specified are considered *unbound*. Unbound actions are invoked through an action import.

Actions whose `IsBound` attribute is `true` are considered *bound*. Bound actions are invoked by appending a segment containing the qualified action name to a segment of the appropriate binding parameter type within the resource path. Bound actions MUST contain at least one `edm:Parameter` element, and the first parameter is the binding parameter. The binding parameter can be of any type, and it MAY be nullable.

12.1.3 Attribute `EntitySetPath`

Bound actions that return an entity or a collection of entities MAY specify a value for the `EntitySetPath` attribute if determination of the entity set for the return type is contingent on the binding parameter.

The value for the `EntitySetPath` attribute consists of a series of segments joined together with forward slashes.

The first segment of the entity set path MUST be the name of the binding parameter. The remaining segments of the entity set path MUST represent navigation segments or type casts.

A navigation segment names the SimpleIdentifier of the navigation property to be traversed. A type cast segment names the QualifiedName of the entity type that should be returned from the type cast.

12.2 Element edm:Function

The edm:Function element represents a function in an entity model.

Functions MUST NOT have observable side effects and MUST return a single instance or a collection of instances of any type. Functions MAY be composable.

The function MUST specify a return type using the edm:ReturnType element. The return type must be a primitive, entity or complex type, or a collection of primitive, entity or complex types.

The function may also define zero or more edm:Parameter elements to be used during the execution of the function.

12.2.1 Attribute Name

The edm:Function element MUST include a Name attribute whose value is a SimpleIdentifier.

12.2.1.1 Function Overload Rules

Bound functions support overloading (multiple functions having the same name within the same namespace) subject to the following rules:

- The combination of function name, binding parameter type, and unordered set of non-binding parameter names MUST be unique within a namespace.
- The combination of function name, binding parameter type, and ordered set of parameter types MUST be unique within a namespace.
- All bound functions with the same function name and binding parameter type within a namespace MUST specify the same return type.

Unbound functions support overloading subject to the following rules:

- The combination of function name and unordered set of parameter names MUST be unique within a namespace.
- The combination of function name and ordered set of parameter types MUST be unique within a namespace.
- All unbound functions with the same function name within a namespace MUST specify the same return type.

An unbound function MAY have the same name as a bound function.

Note that type definitions can be used to disambiguate overloads for both bound and unbound functions, even if they specify the same underlying type.

12.2.2 Attribute IsBound

A function element MAY specify a Boolean value for the IsBound attribute.

Functions whose IsBound attribute is false or not specified are considered *unbound*. Unbound functions are invoked as static functions within a filter or orderby expression, or from the entity container through a function import.

Functions whose IsBound attribute is true are considered *bound*. Bound functions are invoked by appending a segment containing the qualified function name to a segment of the appropriate binding parameter type within a resource path, filter, or orderby expression. Bound functions MUST contain at least one edm:Parameter element, and the first parameter is the binding parameter. The binding parameter can be of any type, and it MAY be nullable.

12.2.3 Attribute IsComposable

A function element MAY specify a Boolean value for the IsComposable attribute. If no value is specified for the IsComposable attribute, the value defaults to false.

Functions whose `IsComposable` attribute is `true` are considered *composable*. A composable function can be invoked with additional path segments or system query options appended to the path that identifies the composable function as appropriate for the type returned by the composable function.

12.2.4 Attribute `EntitySetPath`

Bound functions that return an entity or a collection of entities MAY specify a value for the `EntitySetPath` attribute if determination of the entity set for the return type is contingent on the binding parameter.

The value for the `EntitySetPath` attribute consists of a series of segments joined together with forward slashes.

The first segment of the entity set path MUST be the name of the binding parameter. The remaining segments of the entity set path MUST represent navigation segments or type casts.

A navigation segment names the SimpleIdentifier of the navigation property to be traversed. A type cast segment names the QualifiedName of the entity type that should be returned from the type cast.

12.3 Element `edm:ReturnType`

The attributes `MaxLength`, `Precision`, `Scale`, and `SRID` can be used to specify the facets of the return type, as appropriate. If the facet attributes are not specified, their values are considered unspecified.

12.3.1 Attribute `Type`

The `Type` attribute specifies the type of the result returned by the function or action.

12.3.2 Attribute `Nullable`

A return type MAY specify a Boolean value for the `Nullable` attribute. If not specified, the `Nullable` attribute defaults to `true`.

If the return type has a `Type` attribute that does not specify a collection, the value of `true` means that the action or function may return a single `null` value. A value of `false` means that the action or function will never return a `null` value and instead fail with an error response if it cannot compute a result.

If the return type has a `Type` attribute that specifies a collection, the result will always exist, but the collection MAY be empty. In this case, the `Nullable` attribute applies to members of the collection and specifies whether the collection can contain null values.

12.4 Element `edm:Parameter`

The `edm:Parameter` element allows one or more parameters to be passed to a function or action.

Example 28: a function returning the top-selling products for a given year. In this case the year must be specified as a parameter of the function with the `edm:Parameter` element.

```
<Function Name="TopSellingProducts">
  <Parameter Name="Year" Type="Edm.Decimal" Precision="4" Scale="0" />
  <ReturnType Type="Collection(Model.Product)" />
</Function>
```

12.4.1 Attribute `Name`

The `edm:Parameter` element MUST include a `Name` attribute whose value is a SimpleIdentifier. The parameter name MUST be unique within its parent element.

12.4.2 Attribute `Type`

The `edm:Parameter` element MUST include the `Type` attribute whose value is a TypeName indicating the type of value that can be passed to the parameter.

12.4.3 Attribute `Nullable`

A parameter whose `Type` attribute does not specify a collection MAY specify a Boolean value for the `Nullable` attribute. If not specified, the `Nullable` attribute defaults to `true`.

The value of `true` means that the parameter accepts a `null` value.

12.4.4 Parameter Facets

An `edm:Parameter` element MAY specify values for the `MaxLength`, `Precision`, `Scale`, or `SRID` attributes. The descriptions of these facets and their implications are covered in section 6.2.

13 Entity Container

Each metadata document used to describe an OData service MUST define exactly one entity container. Entity containers define the entity sets, singletons, function and action imports exposed by the service.

An *entity set* allows access to entity type instances. Simple entity models frequently have one entity set per entity type.

Example 29: one entity set per entity type

```
<EntitySet Name="Products"   EntityType="Self.Product" />
<EntitySet Name="Categories" EntityType="Self.Category" />
```

Other entity models may expose multiple entity sets per type.

Example 30: three entity sets referring to the two entity types

```
<EntitySet Name="StandardCustomers" EntityType="Self.Customer">
  <NavigationPropertyBinding Path="Orders" Target="Orders" />
</EntitySet>
<EntitySet Name="PreferredCustomers" EntityType="Self.Customer">
  <NavigationPropertyBinding Path="Orders" Target="Orders" />
</EntitySet>
<EntitySet Name="Orders" EntityType="Self.Order" />
```

There are separate entity sets for standard customers and preferred customers, but only one entity set for orders. The entity sets for standard customers and preferred customers both have navigation property bindings to the orders entity set, but the orders entity set does not have a navigation property binding for the Customer navigation property, since it could lead to either set of customers.

An entity set can expose instances of the specified entity type as well as any entity type inherited from the specified entity type.

A *singleton* allows addressing a single entity directly from the entity container without having to know its key, and without requiring an entity set.

A *function import* or an *action import* is used to expose a function or action defined in an entity model as a top level resource.

Example 31: function import returning the top ten revenue-generating products for a given fiscal year

```
<FunctionImport Name="TopSellingProducts"
                Function="Model.TopSellingProducts"
                EntitySet="Products" />
```

Example 32: An entity container aggregates entity sets, singletons, action imports, and function imports.

```
<EntityContainer Name="DemoService">
  <EntitySet Name="Products" EntityType="Self.Product">
    <NavigationPropertyBinding Path="Category"
                               Target="Self.DemoService.Categories" />
    <NavigationPropertyBinding Path="Supplier"
                               Target="Self.DemoService.Suppliers" />
  </EntitySet>
  <EntitySet Name="Categories" EntityType="Self.Category">
    <NavigationPropertyBinding Path="Products"
                               Target="Self.DemoService.Products" />
  </EntitySet>
  <EntitySet Name="Suppliers" EntityType="Self.Supplier">
    <NavigationPropertyBinding Path="Products"
                               Target="Self.DemoService.Products" />
  </EntitySet>
  <Singleton Name="Contoso" Type="Self.Supplier" />
```

```
  <ActionImport Name="LeaveRequestApproval" Action="Self.Approval" />
  <FunctionImport Name="ProductsByRating" Function="Self.ProductsByRating"
                  EntitySet="Products" />
</EntityContainer>
```

13.1 Element edm:EntityContainer

The edm:EntityContainer element represents an entity container in an entity model. It corresponds to a virtual or physical data store and contains one or more edm:EntitySet, edm:Singleton, edm:ActionImport, or edm:FunctionImport elements. Entity set, singleton, action import, and function import names MUST be unique within an entity container.

13.1.1 Attribute Name

The edm:EntityContainer element MUST provide a unique SimpleIdentifier value for the Name attribute.

13.1.2 Attribute Extends

The edm:EntityContainer element MAY include an Extends attribute whose value is the QualifiedName of an entity container in scope. All children of the "base" entity container specified in the Extends attribute are added to the "extending" entity container that has the Extends attribute.

Example 33: the entity container Extending will contain all child elements that it defines itself, plus all child elements of the Base entity container located in SomeOtherSchema

```
<EntityContainer Name="Extending" Extends="SomeOtherSchema.Base">
   ...
</EntityContainer>
```

13.2 Element edm:EntitySet

The edm:EntitySet element represents an entity set in an entity model.

13.2.1 Attribute Name

The edm:EntitySet element MUST include a Name attribute whose value is a SimpleIdentifier.

13.2.2 Attribute EntityType

The edm:EntitySet element MUST include an EntityType attribute whose value is the QualifiedName of an entity type in scope. Each entity type in the model may have zero or more entity sets that reference the entity type.

An entity set MUST contain only instances of the entity type specified by the EntityType attribute or its subtypes. The entity type named by the EntityType attribute MAY be abstract but MUST have a key defined.

13.2.3 Attribute IncludeInServiceDocument

The edm:EntitySet element MAY include the IncludeInServiceDocument attribute whose Boolean value indicates whether the entity set is advertised in the service document.

If no value is specified for this attribute, its value defaults to true.

Entity sets that cannot be queried without specifying additional query options SHOULD specify the value false for this attribute.

13.3 Element edm:Singleton

The edm:Singleton element represents a single entity in an entity model, called a *singleton.*

13.3.1 Attribute Name

The edm:Singleton element MUST include a Name attribute whose value is a SimpleIdentifier.

13.3.2 Attribute Type

The edm:Singleton element MUST include a Type attribute whose value is the QualifiedName of an entity type in scope. Each entity type in the model may be used in zero or more edm:Singleton elements.

A singleton MUST reference an instance of the entity type specified by the Type attribute.

13.4 Element edm:NavigationPropertyBinding

An entity set or a singleton SHOULD contain an edm:NavigationPropertyBinding element for each navigation property of its entity type, including navigation properties defined on complex typed properties.

If omitted, clients MUST assume that the target entity set or singleton can vary per related entity.

13.4.1 Attribute Path

A navigation property binding MUST name a navigation property of the entity set's, singleton's, or containment navigation property's entity type or one of its subtypes in the Path attribute. If the navigation property is defined on a subtype, the path attribute MUST contain the QualifiedName of the subtype, followed by a forward slash, followed by the navigation property name. If the navigation property is defined on a complex type used in the definition of the entity set's entity type, the path attribute MUST contain a forward-slash separated list of complex property names and qualified type names that describe the path leading to the navigation property.

The path can traverse one or more containment navigation properties but the last segment MUST be a non-containment navigation property and there MUST NOT be any non-containment navigation properties prior to the final segment.

A navigation property MUST NOT be named in more than one navigation property binding; navigation property bindings are only used when all related entities are known to come from a single entity set.

13.4.2 Attribute Target

A navigation property binding MUST specify a SimpleIdentifier or TargetPath value for the Target attribute that specifies the entity set, singleton, or containment navigation property that contains the related instance(s) targeted by the navigation property specified in the Path attribute.

If the value of the Target attribute is a SimpleIdentifier, it MUST resolve to an entity set or singleton defined in the same entity container as the enclosing element.

If the value of the Target attribute is a TargetPath, it MUST resolve to an entity set, singleton, or containment navigation property in scope. The path can traverse containment navigation properties or complex properties before ending in a containment navigation property, but there MUST not be any non-containment navigation properties prior to the final segment.

Example 34: for an entity set in the same container as the enclosing entity set Categories

```
<EntitySet Name="Categories" EntityType="Self.Category">
  <NavigationPropertyBinding Path="Products"
                             Target="SomeSet" />
</EntitySet>
```

Example 35: for an entity set in any container in scope


```
<EntitySet Name="Categories" EntityType="Self.Category">
  <NavigationPropertyBinding Path="Products"
                             Target="SomeModel.SomeContainer/SomeSet" />
</EntitySet>
```

13.5 Element edm:ActionImport

The edm:ActionImport element allows exposing an unbound action as a top-level element in an entity container. Action imports are never advertised in the service document.

13.5.1 Attribute Name

The edm:ActionImport element MUST include a Name attribute whose value is a SimpleIdentifier. It MAY be identical to the last segment of the QualifiedName used to specify the Action attribute value.

13.5.2 Attribute Action

The edm:ActionImport element MUST include a QualifiedName value for the Action attribute which MUST resolve to the name of an unbound edm:Action element in scope.

13.5.3 Attribute EntitySet

If the return type of the action specified in the Action attribute is an entity or a collection of entities, a SimpleIdentifier or TargetPath value MAY be specified for the EntitySet attribute that names the entity set to which the returned entities belong. If a SimpleIdentifier is specified, it MUST resolve to an entity set defined in the same entity container. If a TargetPath is specified, it MUST resolve to an entity set in scope.

If the return type is not an entity or a collection of entities, a value MUST NOT be defined for the EntitySet attribute.

13.6 Element edm:FunctionImport

The edm:FunctionImport element allows exposing an unbound function as a top-level element in an entity container. All unbound overloads of an imported function can be invoked from the entity container.

13.6.1 Attribute Name

The edm:FunctionImport element MUST include a Name attribute whose value is a SimpleIdentifier. It MAY be identical to the last segment of the QualifiedName used to specify the Function attribute value.

13.6.2 Attribute Function

The edm:FunctionImport element MUST include the Function attribute whose value MUST be a QualifiedName that resolves to the name of an unbound edm:Function element in scope.

13.6.3 Attribute EntitySet

If the return type of the function specified in the Function attribute is an entity or a collection of entities, a SimpleIdentifier or TargetPath value MAY be defined for the EntitySet attribute that names the entity set to which the returned entities belong. If a SimpleIdentifier is specified, it MUST resolve to an entity set defined in the same entity container. If a TargetPath is specified, it MUST resolve to an entity set in scope.

If the return type is not an entity or a collection of entities, a value MUST NOT be defined for the EntitySet attribute.

13.6.4 Attribute `IncludeInServiceDocument`

The `edm:FunctionImport` for a parameterless function MAY include the `IncludeInServiceDocument` attribute whose Boolean value indicates whether the function import is advertised in the service document.

If no value is specified for this attribute, its value defaults to `false`.

14 Vocabulary and Annotation

Vocabularies and annotations provide the ability to annotate metadata as well as instance data, and define a powerful extensibility point for OData. An *annotation* applies a *term* to a model element and defines how to calculate a value for the applied term.

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 may define ranges of valid values for a particular property. Metadata annotations are applied in CSDL documents describing or referencing an entity model.

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 the same annotation is defined at both the metadata and instance level, the instance-level annotation overrides the annotation specified at the metadata level. Instance annotations appear in the actual payload as described in **[OData-Atom]** and **[OData-JSON]**. Annotations that apply across instances should be specified as metadata annotations.

A *vocabulary* is a namespace containing a set of terms where each term is a named metadata extension. Anyone can define a vocabulary (a set of terms) that is scenario-specific or company-specific; more commonly used terms can be published as shared vocabularies such as the OData Core vocabulary **[OData-VocCore]**.

A term can be used:

- To extend model elements and type instances with additional information.
- To map instances of annotated structured types to an interface defined by the term type; i.e. annotations allow viewing instances of a structured type as instances of a differently structured type specified by the applied term.

A service SHOULD NOT require a client to interpret annotations.

Example 36: the `Product` entity type is extended with a `DisplayName` by a metadata annotation that binds the term `DisplayName` to the value of the property `Name`. The `Product` entity type also includes an annotation that allows its instances to be viewed as instances of the type specified by the term `SearchResult`

```
<EntityType Name="Product">
  <Key>
    <PropertyRef Name="ID" />
  </Key>
  <Property Name="ID" Nullable="false" Type="Edm.Int32" />
  <Property Name="Name" Type="Edm.String" />
  <Property Name="Description" Type="Edm.String" />
  ...
  <Annotation Term="UI.DisplayName" Path="Name" />
  <Annotation Term="SearchVocabulary.SearchResult">
    <Record>
      <PropertyValue Property="Title" Path="Name" />
      <PropertyValue Property="Abstract" Path="Description" />
      <PropertyValue Property="Url">
        <Apply Function="odata.concat">
          <String>Products(</String>
          <Path>ID</Path>
          <String>)</String>
        </Apply>
      </PropertyValue>
    </Record>
  </Annotation>
</EntityType>
```

14.1 Element edm:Term

The edm:Term element defines a term in a vocabulary.

A term allows annotating a CSDL element or OData resource representation with additional data.

14.1.1 Attribute Name

The edm:Term element MUST include a Name attribute whose value is a SimpleIdentifier.

14.1.2 Attribute Type

The edm:Term element MUST include a Type attribute whose value is a TypeName. It indicates what type of value must be returned by the expression contained in an annotation using the term.

14.1.3 Attribute BaseTerm

The edm:Term element MAY provide a QualifiedName value for the BaseTerm attribute. The value of the BaseTerm attribute MUST be the name of a term in scope. When applying a term with a base term, the base term MUST also be applied with the same qualifier, and so on until a term without a base term is reached.

14.1.4 Attribute DefaultValue

A edm:Term element whose Type attribute specifies a primitive or enumeration type MAY define a value for the DefaultValue attribute. The value of this attribute determines the value of the term when applied in an edm:Annotation without providing an expression.

Default values of type Edm.String MUST be represented according to the XML escaping rules for character data in attribute values. Values of other primitive types MUST be represented according to the appropriate alternative in the primitiveValue rule defined in **[OData-ABNF]**, i.e. Edm.Binary as binaryValue, Edm.Boolean as booleanValue etc.

If no value is specified, the DefaultValue attribute defaults to null.

14.1.5 Attribute AppliesTo

The edm:Term element MAY define a value for the AppliesTo attribute. The value of this attribute is a whitespace-separated list of CSDL element names that this term is intended to be applied to. If no value is supplied, the term is not intended to be restricted in its application. As the intended usage may evolve over time, clients SHOULD be prepared for any annotation to be applied to any element.

*Example 37: the IsURI term can be applied to properties and terms that are of type Edm.String (the Core.Tag type and the two Core terms are defined in **[OData-VocCore]**)*

```
<Term Name="IsURI" Type="Core.Tag" DefaultValue="true"
      AppliesTo="Property">
  <Annotation Term="Core.Description">
    <String>
      Properties and terms annotated with this term MUST contain a valid URI
    </String>
  </Annotation>
  <Annotation Term="Core.RequiresType" String="Edm.String" />
</Term>
```

14.1.6 Term Facets

The edm:Term element MAY specify values for the Nullable, DefaultValue, MaxLength, Precision, Scale, or SRID attributes. These facets and their implications are described in section 6.2.

14.2 Element edm:Annotations

The edm:Annotations element is used to apply a group of annotations to a single model element. It MUST contain at least one edm:Annotation element.

14.2.1 Attribute Target

The edm:Annotations element MUST include a Target attribute whose value is a path expression that MUST resolve to a model element in the entity model.

External targeting is only possible for EDM elements that are uniquely identified within their parent, and all their ancestor elements are uniquely identified within their parent:

- edm:ActionImport
- edm:ComplexType
- edm:EntityContainer
- edm:EntitySet
- edm:EntityType
- edm:EnumType
- edm:FunctionImport
- edm:Member
- edm:NavigationProperty
- edm:Property
- edm:Singleton
- edm:Term
- edm:TypeDefinition

These are the direct children of a schema with a unique name (i.e. except actions and functions whose overloads to not possess a natural identifier), and all direct children of an entity container. The edm:Schema element and most of the not uniquely identifiable EDM elements can still be annotated using an inline edm:Annotation element.

External targeting is possible for actions, functions, and their parameters, in which case the annotation applies to all overloads of the action or function or all parameters of that name across all overloads. External targeting of individual action or function overloads is not possible.

External targeting is also possible for properties and navigation properties of singletons or entities in a particular entity set. These annotations override annotations on the properties or navigation properties targeted via the declaring structured type.

The allowed path expressions are:

- QualifiedName of schema child
- QualifiedName of schema child followed by a forward slash and name of child element
- QualifiedName of an entity container followed by a segment containing a singleton or entity set name and zero or more property, navigation property, or type cast segments

Example 38: Target expressions

```
MySchema.MyEntityType
MySchema.MyEntityType/MyProperty
MySchema.MyEntityType/MyNavigationProperty
MySchema.MyComplexType
MySchema.MyComplexType/MyProperty
MySchema.MyComplexType/MyNavigationProperty
MySchema.MyEnumType
MySchema.MyEnumType/MyMember
MySchema.MyTypeDefinition
MySchema.MyTerm
MySchema.MyEntityContainer
MySchema.MyEntityContainer/MyEntitySet
MySchema.MyEntityContainer/MySingleton
MySchema.MyEntityContainer/MyActionImport
MySchema.MyEntityContainer/MyFunctionImport
MySchema.MyAction
MySchema.MyFunction
```



```
MySchema.MyFunction/MyParameter
MySchema.MyEntityContainer/MyEntitySet/MyProperty
MySchema.MyEntityContainer/MyEntitySet/MyNavigationProperty
MySchema.MyEntityContainer/MyEntitySet/MySchema.MyEntityType/MyProperty
MySchema.MyEntityContainer/MyEntitySet/MySchema.MyEntityType/MyNavProperty
MySchema.MyEntityContainer/MyEntitySet/MyComplexProperty/MyProperty
MySchema.MyEntityContainer/MyEntitySet/MyComplexProperty/MyNavigationProperty
MySchema.MyEntityContainer/MySingleton/MyComplexProperty/MyNavigationProperty
```

14.2.2 Attribute `Qualifier`

An `edm:Annotations` element MAY provide a SimpleIdentifier value for the `Qualifier` attribute.

The `Qualifier` attribute allows annotation authors a means of conditionally applying an annotation.

Example 39: annotations should only be applied to tablet devices

```
<Annotations Target="Self.Person" Qualifier="Tablet">
  ...
</Annotations>
```

14.3 Element `edm:Annotation`

The `edm:Annotation` element represents a single annotation. An annotation applies a term to a model element and defines how to calculate a value for the term application. The following model elements MAY be annotated with a term:

- `edm:Action`
- `edm:ActionImport`
- `edm:Annotation`
- `edm:Apply`
- `edm:Cast`
- `edm:ComplexType`
- `edm:EntityContainer`
- `edm:EntitySet`
- `edm:EntityType`
- `edm:EnumType`
- `edm:Function`
- `edm:FunctionImport`
- `edm:If`
- `edm:IsOf`
- `edm:LabeledElement`
- `edm:Member`
- `edm:NavigationProperty`
- `edm:Null`
- `edm:OnDelete`
- `edm:Parameter`
- `edm:Property`
- `edm:PropertyValue`
- `edm:Record`
- `edm:ReferentialConstraint`
- `edm:ReturnType`
- `edm:Schema`
- `edm:Singleton`
- `edm:Term`
- `edm:TypeDefinition`
- `edm:UrlRef`
- `edmx:Reference`
- all Comparison and Logical Operators

An `edm:Annotation` element can be used as a child of the model element it annotates, or as the child of an `edm:Annotations` element that targets the model element to be annotated.

An `edm:Annotation` element MAY contain a constant expression or dynamic expression in either attribute or element notation. If no expression is specified for a term with a primitive type, the annotation evaluates to the default value of the term definition. If no expression is specified for a term with a complex type, the annotation evaluates to a complex instance with default values for all properties is used. If no expression is specified for a collection-valued term, the annotation evaluates to an empty collection.

If an entity type or complex type is annotated with a term that itself has a structured type, an instance of the annotated type may be viewed as an “instance” of the term, and the qualified term name may be used as a term-cast segment in path expressions.

14.3.1 Attribute `Term`

An annotation element MUST provide a QualifiedName value for the `Term` attribute. The value of the `Term` attribute MUST be the name of a term in scope. The target of the annotation MUST comply with any `AppliesTo` constraint.

14.3.2 Attribute `Qualifier`

An annotation element MAY provide a SimpleIdentifier value for the `Qualifier` attribute.

The qualifier attribute allows annotation authors a means of conditionally applying an annotation.

Example 40: annotation should only be applied to tablet devices

```
<Annotation Term="org.example.display.DisplayName" Path="FirstName"
            Qualifier="Tablet" />
```

Annotation elements that are children of an `edm:Annotations` element MUST NOT provide a value for the qualifier attribute if the parent `edm:Annotations` element provides a value for the qualifier attribute.

14.4 Constant Expressions

Constant expressions allow assigning a constant value to an applied term. The constant expressions support element and attribute notation.

Example 41: two annotations intended as user interface hints

```
<EntitySet Name="Products" EntityType="Self.Product">
  <Annotation Term="org.example.display.DisplayName"
              String="Product Catalog" />
</EntitySet>

<EntitySet Name="Suppliers" EntityType="Self.Supplier">
  <Annotation Term="org.example.display.DisplayName">
    <String>Supplier Directory</String>
  </Annotation>
</EntitySet>
```

14.4.1 Expression `edm:Binary`

The `edm:Binary` expression evaluates to a primitive binary value. A binary expression MUST be assigned a value conforming to the rule `binaryValue` in **[OData-ABNF]**.

The binary expression MAY be provided using element notation or attribute notation.

Example 42: base64url-encoded binary value (OData)

```
<Annotation Term="org.example.display.Thumbnail" Binary="T0RhdGE" />

<Annotation Term="org.example.display.Thumbnail">
  <Binary>T0RhdGE</Binary>
</Annotation>
```

14.4.2 Expression `edm:Bool`

The `edm:Bool` expression evaluates to a primitive Boolean value. A Boolean expression MUST be assigned a Boolean value.

The Boolean expression MAY be provided using element notation or attribute notation.

Example 43:

```
<Annotation Term="org.example.display.ReadOnly" Bool="true" />

<Annotation Term="org.example.display.ReadOnly">
  <Bool>true</Bool>
</Annotation>
```

14.4.3 Expression `edm:Date`

The `edm:Date` expression evaluates to a primitive date value. A date expression MUST be assigned a value of type `xs:date`, see **[XML-Schema-2]**, section 3.3.9. The value MUST also conform to rule `dateValue` in **[OData-ABNF]**, i.e. it MUST NOT contain a time-zone offset.

The date expression MAY be provided using element notation or attribute notation.

Example 44:

```
<Annotation Term="org.example.vCard.birthDay" Date="2000-01-01" />

<Annotation Term="org.example.vCard.birthDay">
  <Date>2000-01-01</Date>
</Annotation>
```

14.4.4 Expression `edm:DateTimeOffset`

The `edm:DateTimeOffset` expression evaluates to a primitive date/time value with a time-zone offset. A date/time expression MUST be assigned a value of type `xs:dateTimeStamp`, see **[XML-Schema-2]**, section 3.4.28. The value MUST also conform to rule `dateTimeOffsetValue` in **[OData-ABNF]**, i.e. it MUST NOT contain an end-of-day fragment (24:00:00).

The date/time expression MAY be provided using element notation or attribute notation.

Example 45:

```
<Annotation Term="org.example.display.LastUpdated"
            DateTimeOffset="2000-01-01T16:00:00.000Z" />

<Annotation Term="org.example.display.LastUpdated">
  <DateTimeOffset>2000-01-01T16:00:00.000-09:00</DateTimeOffset>
</Annotation>
```

14.4.5 Expression `edm:Decimal`

The `edm:Decimal` expression evaluates to a primitive decimal value. A decimal expression MUST be assigned a value conforming to the rule `decimalValue` in **[OData-ABNF]**.

The decimal expression MAY be provided using element notation or attribute notation.

Example 46:

```
<Annotation Term="org.example.display.Width" Decimal="3.14" />

<Annotation Term="org.example.display.Width">
  <Decimal>3.14</Decimal>
</Annotation>
```

14.4.6 Expression edm:Duration

The edm:Duration expression evaluates to a primitive duration value. A duration expression MUST be assigned a value of type xs:dayTimeDuration, see **[XML-Schema-2]**, section 3.4.27.

The duration expression MAY be provided using element notation or attribute notation.

Example 47:

```
<Annotation Term="org.example.task.duration" Duration="P7D" />

<Annotation Term="org.example.task.duration">
  <Duration>P11D23H59M59.999999999999S</Duration>
</Annotation>
```

14.4.7 Expression edm:EnumMember

The edm:EnumMember expression references a member of an enumeration type. An enumeration member expression MUST be assigned a value that consists of the qualified name of the enumeration type, followed by a forward slash and the name of the enumeration member. If the enumeration type specifies an IsFlags attribute with value true, the expression MAY also be assigned a whitespace-separated list of values. Each of these values MUST resolve to the name of a member of the enumeration type of the specified term.

The enumeration member expression MAY be provided using element notation or attribute notation.

Example 48: single value

```
<Annotation Term="org.example.HasPattern"
             EnumMember="org.example.Pattern/Red" />

<Annotation Term="org.example.HasPattern">
  <EnumMember>org.example.Pattern/Red</EnumMember>
</Annotation>
```

Example 49: combined value for IsFlags enumeration type

```
<Annotation Term="org.example.HasPattern"
           EnumMember="org.example.Pattern/Red org.example.Pattern/Striped" />

<Annotation Term="org.example.HasPattern">
  <EnumMember>org.example.Pattern/Red org.example.Pattern/Striped</EnumMember>
</Annotation>
```

14.4.8 Expression edm:Float

The edm:Float expression evaluates to a primitive floating point (or double) value. A float expression MUST be assigned a value conforming to the rule doubleValue in **[OData-ABNF]**.

The float expression MAY be provided using element notation or attribute notation.

Example 50:

```
<Annotation Term="org.example.display.Width" Float="3.14" />

<Annotation Term="org.example.display.Width">
  <Float>3.14</Float>
</Annotation>
```

14.4.9 Expression edm:Guid

The edm:Guid expression evaluates to a primitive 32-character string value. A guid expression MUST be assigned a value conforming to the rule guidValue in **[OData-ABNF]**.

The guid expression MAY be provided using element notation or attribute notation.

Example 51:

```
<Annotation Term="org.example.display.Id"
            Guid="21EC2020-3AEA-1069-A2DD-08002B30309D" />

<Annotation Term="org.example.display.Id">
  <Guid>21EC2020-3AEA-1069-A2DD-08002B30309D</Guid>
</Annotation>
```

14.4.10 Expression edm:Int

The `edm:Int` expression evaluates to a primitive integer value. An integer MUST be assigned a value conforming to the rule `int64Value` in **[OData-ABNF]**.

The integer expression MAY be provided using element notation or attribute notation.

Example 52:

```
<Annotation Term="org.example.display.Width" Int="42" />

<Annotation Term="org.example.display.Width">
  <Int>42</Int>
</Annotation>
```

14.4.11 Expression edm:String

The `edm:String` expression evaluates to a primitive string value. A string expression MUST be assigned a value of the type `xs:string`, see **[XML-Schema-2]**, section 3.3.1.

The string expression MAY be provided using element notation or attribute notation.

Example 53:

```
<Annotation Term="org.example.display.DisplayName" String="Product Catalog" />

<Annotation Term="org.example.display.DisplayName">
  <String>Product Catalog</String>
</Annotation>
```

14.4.12 Expression edm:TimeOfDay

The `edm:TimeOfDay` expression evaluates to a primitive time value. A time-of-day expression MUST be assigned a value conforming to the rule `timeOfDayValue` in **[OData-ABNF]**.

The time-of-day expression MAY be provided using element notation or attribute notation.

Example 54:

```
<Annotation Term="org.example.display.EndTime" TimeOfDay="21:45:00" />

<Annotation Term="org.example.display.EndTime">
  <TimeOfDay>21:45:00</TimeOfDay>
</Annotation>
```

14.5 Dynamic Expressions

Dynamic expressions allow assigning a calculated value to an applied term. The dynamic expressions `edm:AnnotationPath`, `edm:NavigationPropertyPath`, `edm:Path`, `edm:PropertyPath`, and `edm:UrlRef` expressions support element and attribute notation, all other dynamic expressions only support element notation.

14.5.1 Comparison and Logical Operators

The following EDM elements allow service authors to supply a dynamic conditional expression which evaluates to a value of type `Edm.Boolean`. They MAY be combined and they MAY be used anywhere instead of an `edm:Bool` expression.

Element	Description	Example
Logical Operators		
`edm:And`	Logical and	`<And><Path>IsMale</Path><Path>IsMarried</Path></And>`
`edm:Or`	Logical or	`<Or><Path>IsMale</Path><Path>IsMarried</Path></Or>`
`edm:Not`	Logical negation	`<Not><Path>IsMale</Path></Not>`
Comparison Operators		
`edm:Eq`	Equal	`<Eq><Null/><Path>IsMale</Path></Eq>`
`edm:Ne`	Not equal	`<Ne><Null/><Path>IsMale</Path></Ne>`
`edm:Gt`	Greater than	`<Gt><Path>Price</Path><Int>20</Int></Gt>`
`edm:Ge`	Greater than or equal	`<Ge><Path>Price</Path><Int>10</Int></Ge>`
`edm:Lt`	Less than	`<Lt><Path>Price</Path><Int>20</Int></Lt>`
`edm:Le`	Less than or equal	`<Le><Path>Price</Path><Int>100</Int></Le>`

The `edm:And` and `edm:Or` elements require two child expressions that evaluate to Boolean values. The `edm:Not` elements requires a single child expression that evaluates to a Boolean value. For details on null handling for comparison operators see **[OData-URL]**.

The other elements representing the comparison operators require two child expressions that evaluate to comparable values.

14.5.2 Expression `edm:AnnotationPath`

The `edm:AnnotationPath` expression provides a value for terms or term properties that specify the built-in abstract type `Edm.AnnotationPath`. It uses the same syntax and rules as the `edm:Path` expression, with the added restriction that the last path segment MUST be a term cast with optional qualifier in the context of the preceding path part.

In contrast to the `edm:Path` expression the value of the `edm:AnnotationPath` expression is the path itself, not the value of the annotation identified by the path. This is useful for terms that reuse or refer to other terms.

The `edm:AnnotationPath` expression MAY be provided using element notation or attribute notation.

Example 55:

```
<Annotation Term="UI.ReferenceFacet"
            AnnotationPath="Product/Supplier/@UI.LineItem" />

<Annotation Term="UI.CollectionFacet" Qualifier="Contacts">
  <Collection>
    <AnnotationPath>Supplier/@Communication.Contact</AnnotationPath>
    <AnnotationPath>Customer/@Communication.Contact</AnnotationPath>
  </Collection>
</Annotation>
```

14.5.3 Expression edm:Apply

The edm:Apply expression enables a value to be obtained by applying a client-side function. The Apply expression MUST contain at least one expression. The expressions contained within the Apply expression are used as parameters to the function. The edm:Apply expression MUST be written with element notation.

14.5.3.1 Attribute Function

The edm:Apply expression MUST include a Function attribute whose value is a QualifiedName specifying the name of the client-side function to apply.

OData defines the following canonical functions. Services MAY support additional functions that MUST be qualified with a namespace or alias other than odata. Function names qualified with odata are reserved for this specification and its future versions.

14.5.3.1.1 Function odata.concat

The odata.concat standard client-side function takes two or more expressions as arguments. Each argument MUST evaluate to a primitive or enumeration type. It returns a value of type Edm.String that is the concatenation of the literal representations of the results of the argument expressions. Values of primitive types other than Edm.String are represented according to the appropriate alternative in the primitiveValue rule of **[OData-ABNF]**, i.e. Edm.Binary as binaryValue, Edm.Boolean as booleanValue etc.

Example 56:

```
<Annotation Term="org.example.display.DisplayName">
  <Apply Function="odata.concat">
    <String>Product: </String>
    <Path>ProductName</Path>
    <String> (</String>
    <Path>Available/Quantity</Path>
    <String> </String>
    <Path>Available/Unit</Path>
    <String> available)</String>
  </Apply>
</Annotation>
```

ProductName is of type String, Quantity in complex type Available is of type Decimal, and Unit in Available is of type enumeration, so the result of the Path expression is represented as the member name of the enumeration value.

14.5.3.1.2 Function odata.fillUriTemplate

The odata.fillUriTemplate standard client-side function takes two or more expressions as arguments and returns a value of type Edm.String.

The first argument MUST be of type Edm.String and specifies a URI template according to **[RFC6570]**, the other arguments MUST be edm:LabeledElement expressions. Each edm:LabeledElement expression specifies the template parameter name in its Name attribute and evaluates to the template parameter value.

[RFC6570] defines three kinds of template parameters: simple values, lists of values, and key-value maps.

Simple values are represented as edm:LabeledElement expressions that evaluate to a single primitive value. The literal representation of this value according to **[OData-ABNF]** is used to fill the corresponding template parameter.

Lists of values are represented as edm:LabeledElement expressions that evaluate to a collection of primitive values.

Key-value maps are represented as `edm:LabeledElement` expressions that evaluate to a collection of complex types with two properties that are used in lexicographic order. The first property is used as key, the second property as value.

Example 57: assuming there are no special characters in values of the `NameOfMovieGenre` property

```
<Apply Function="odata.fillUriTemplate">
  <String>http://host/service/Genres('{genreName}')</String>
  <LabeledElement Name="genreName" Path="NameOfMovieGenre" />
</Apply>
```

14.5.3.1.3 Function `odata.uriEncode`

The `odata.uriEncode` standard client-side function takes one argument of primitive type and returns the URL-encoded OData literal that can be used as a key value in OData URLs or in the query part of OData URLs. Note: string literals are surrounded by single quotes.

Example 58:

```
<Apply Function="odata.fillUriTemplate">
  <String>http://host/service/Genres({genreName})</String>
  <LabeledElement Name="genreName">
    <Apply Function="odata.uriEncode" >
      <Path>NameOfMovieGenre</Path>
    </Apply>
  </LabeledElement>
</Apply>
```

14.5.4 Expression `edm:Cast`

The `edm:Cast` expression casts the value obtained from its single child expression to the specified type. The cast expression follows the same rules as the `cast` canonical function defined in **[OData-URL]**.

The cast expression MUST specify a `Type` attribute and contain exactly one expression.

The cast expression MUST be written with element notation.

Example 59:

```
<Annotation Term="org.example.display.Threshold">
  <Cast Type="Edm.Decimal">
    <Path>Average</Path>
  </Cast>
</Annotation>
```

14.5.4.1 Attribute `Type`

The `edm:Cast` expression MUST specify a `Type` attribute whose value is a TypeName in scope.

If the specified type is a primitive type, the facet attributes `MaxLength`, `Precision`, `Scale`, and `SRID` MAY be specified if applicable to the specified primitive type. If the facet attributes are not specified, their values are considered unspecified.

14.5.5 Expression `edm:Collection`

The `edm:Collection` expression enables a value to be obtained from zero or more child expressions. The value calculated by the collection expression is the collection of the values calculated by each of the child expressions.

The collection expression contains zero or more child expressions. The values of the child expressions MUST all be type compatible.

The collection expression MUST be written with element notation.

Example 60:

```
<Annotation Term="org.example.seo.SeoTerms">
  <Collection>
    <String>Product</String>
    <String>Supplier</String>
    <String>Customer</String>
  </Collection>
</Annotation>
```

14.5.6 Expression `edm:If`

The `edm:If` expression enables a value to be obtained by evaluating a *conditional expression*. It MUST contain exactly three child elements with dynamic or static expressions. There is one exception to this rule: if and only if the `edm:If` expression is a direct child of `edm:Collection` element the third child element MAY be omitted (this can be used to conditionally add an element to a collection).

The first child element is the conditional expression and MUST evaluate to a Boolean result, e.g. the comparison and logical operators can be used.

The second and third child elements are the expressions, which are evaluated conditionally. The result MUST be type compatible with the type expected by the surrounding element or expression.

If the first expression evaluates to `true`, the second child element MUST be evaluated and its value MUST be returned as the result of the `edm:If` expression. If the conditional expression evaluates to `false` and a third child element is present, it MUST be evaluated and its value MUST be returned as the result of the `edm:If` expression. If no third child element is present, nothing is added to the collection.

The `edm:If` expression MUST be written with element notation, as shown in the following example.

Example 61:

```
<Annotation Term="org.example.person.Gender">
  <If>
    <Path>IsFemale</Path>
    <String>Female</String>
    <String>Male</String>
  </If>
</Annotation>
```

14.5.7 Expression `edm:IsOf`

The `edm:IsOf` expression evaluates a child expression and returns a Boolean value indicating whether the child expression returns the specified type.

An `edm:IsOf` expression MUST specify a `Type` attribute and contain exactly one child expression. The `edm:IsOf` expression MUST return `true` if the child expression returns a type that is compatible with the type named in the `Type` attribute. The `edm:IsOf` expression MUST return `false` if the child expression returns a type that is not compatible with the type named in the `Type` attribute.

The `edm:IsOf` expression MUST be written with element notation.

Example 62:

```
<Annotation Term="Self.IsPreferredCustomer">
  <IsOf Type="Self.PreferredCustomer">
    <Path>Customer</Path>
  </IsOf>
</Annotation>
```

14.5.7.1 Attribute `Type`

The `edm:IsOf` expression MUST specify a `Type` attribute whose value is a TypeName in scope.

If the specified type is a primitive type, the facet attributes `MaxLength`, `Precision`, `Scale`, and `SRID` MAY be specified if applicable to the specified primitive type. If the facet attributes are not specified, their values are considered unspecified.

14.5.8 Expression `edm:LabeledElement`

The `edm:LabeledElement` expression assigns a name to a child expression. The value of the child expression can then be reused elsewhere with an `edm:LabeledElementReference` expression.

A labeled-element expression MUST contain exactly one child expression written either in attribute notation or element notation. The value of the child expression is passed through the labeled-element expression.

A labeled-element expression MUST be written with element notation.

Example 63:

```
<Annotation Term="org.example.display.DisplayName">
  <LabeledElement Name="CustomerFirstName" Path="FirstName" />
</Annotation>

<Annotation Term="org.example.display.DisplayName">
  <LabeledElement Name="CustomerFirstName">
    <Path>FirstName</Path>
  </LabeledElement>
</Annotation>
```

14.5.8.1 Attribute `Name`

An `edm:LabeledElement` expression MUST provide a SimpleIdentifier value for the `Name` attribute that is unique within the schema containing the expression.

14.5.9 Expression `edm:LabeledElementReference`

The `edm:LabeledElementReference` expression returns the value of an `edm:LabeledElement` expression.

The labeled-element reference expression MUST contain the QualifiedName name of a labeled element expression in scope.

The labeled-element reference expression MUST be written with element notation.

Example 64:

```
<Annotation Term="org.example.display.DisplayName">
  <LabeledElementReference>Model.CustomerFirstName</LabeledElementReference>
</Annotation>
```

14.5.10 Expression `edm:Null`

The `edm:Null` expression returns an untyped null value. The null expression MUST NOT contain any other elements or expressions.

The null expression MUST be written with element notation.

Example 65:

```
<Annotation Term="org.example.display.DisplayName">
  <Null/>
</Annotation>
```


14.5.11 Expression edm:NavigationPropertyPath

The edm:NavigationPropertyPath expression provides a value for terms or term properties that specify the built-in abstract type Edm.NavigationPropertyPath. It uses the same syntax and rules as the edm:Path expression with the following exceptions:

- The NavigationPropertyPath expression may traverse multiple collection-valued structural or navigation properties
- The last path segment MUST resolve to a navigation property in the context of the preceding path part, or to a term cast where the term MUST be of type Edm.EntityType, a concrete entity type or a collection of Edm.EntityType or concrete entity type.

In contrast to the edm:Path expression, the value of the edm:NavigationPropertyPath expression is the path itself, not the target instance(s) of the navigation property identified by the path.

The edm:NavigationPropertyPath expression MAY be provided using element notation or attribute notation.

Example 66:

```
<Annotation Term="UI.HyperLink" NavigationPropertyPath="Supplier" />

<Annotation Term="Capabilities.UpdateRestrictions">
  <Record>
    <PropertyValue Property="NonUpdatableNavigationProperties">
      <Collection>
        <NavigationPropertyPath>Supplier</NavigationPropertyPath>
        <NavigationPropertyPath>Category</NavigationPropertyPath>
      </Collection>
    </PropertyValue>
  </Record>
</Annotation>
```

14.5.12 Expression edm:Path

The edm:Path expression enables a value to be obtained by traversing an object graph. It can be used in annotations that target entity containers, entity sets, entity types, complex types, navigation properties of structured types, and properties of structured types.

The value assigned to the path expression MUST be composed of zero or more path segments joined together by forward slashes (/).

If a path segment is a QualifiedName, it represents a *type cast*, and the segment MUST be the name of a type in scope. If the instance identified by the preceding path part cannot be cast to the specified type, the path expression evaluates to the null value.

If a path segment starts with an at (@) character, it represents a *term cast*. The at (@) character MUST be followed by a QualifiedName that MAY be followed by a hash (#) character and a SimpleIdentifier. The QualifiedName preceding the hash character MUST resolve to a term that is in scope, the SimpleIdentifier following the hash sign is interpreted as a Qualifier for the term. If the instance identified by the preceding path part has been annotated with that term (and if present, with that qualifier), the term cast evaluates to the value of that annotation, otherwise it evaluates to the null value. Three special terms are implicitly "annotated" for media entities and stream properties:

- odata.mediaEditLink
- odata.mediaReadLink
- odata.mediaContentType

If a path segment is a SimpleIdentifier, it MUST be the name of a structural property or a navigation property of the instance identified by the preceding path part.

When used within an `edm:Path` expression, a path may contain at most one segment representing a multi-valued structural or navigation property. The result of the expression is the collection of instances resulting from applying the remaining path to each instance in the multi-valued property.

A path may terminate in a `$count` segment if the previous segment is multi-valued, in which case the path evaluates to the number of elements identified by the preceding segment.

If a path segment starts with a navigation property followed by an at (`@`) character, then the at (`@`) character MUST be followed by a QualifiedName that MAY be followed by a hash (`#`) character and a SimpleIdentifier. The QualifiedName preceding the hash character MUST resolve to a term that is in scope, the SimpleIdentifier following the hash sign is interpreted as a Qualifier for the term. If the navigation property has been annotated with that term (and if present, with that qualifier), the path segment evaluates to the value of that annotation, otherwise it evaluates to the null value.

Annotations MAY be embedded within their target, or embedded within an `edm:Annotations` element that specifies the annotation target with a path expression in its `Target` attribute. The latter situation is referred to as *targeting* in the remainder of this section.

For annotations embedded within or targeting an entity container, the path expression is evaluated starting at the entity container, i.e. an empty path resolves to the entity container, and non-empty path values MUST start with the name of a container child (entity set, function import, action import, or singleton). The subsequent segments follow the rules for path expressions targeting the corresponding child element.

For annotations embedded within or targeting an entity set or a singleton, the path expression is evaluated starting at the entity set, i.e. an empty path resolves to the entity set, and non-empty paths MUST follow the rules for annotations targeting the declared entity type of the entity set or singleton.

For annotations embedded within or targeting an entity type or complex type, the path expression is evaluated starting at the type, i.e. an empty path resolves to the type, and the first segment of a non-empty path MUST be a property or navigation property of the type, a type cast, or a term cast.

For annotations embedded within a property of an entity type or complex type, the path expression is evaluated starting at the directly enclosing type. This allows e.g. specifying the value of an annotation on one property to be calculated from values of other properties of the same type. An empty path resolves to the enclosing type, and non-empty paths MUST follow the rules for annotations targeting the directly enclosing type.

For annotations targeting a property of an entity type or complex type, the path expression is evaluated starting at the *outermost* entity type or complex type named in the `Target` of the enclosing `edm:Annotations` element, i.e. an empty path resolves to the outermost type, and the first segment of a non-empty path MUST be a property or navigation property of the outermost type, a type cast, or a term cast.

A path expression MAY be provided using element notation or attribute notation.

Example 67:

```
<Annotation Term="org.example.display.DisplayName" Path="FirstName" />

<Annotation Term="org.example.display.DisplayName">
  <Path>@vCard.Address#work/FullName</Path>
</Annotation>
```

14.5.13 Expression `edm:PropertyPath`

The `edm:PropertyPath` expression provides a value for terms or term properties that specify the built-in abstract type `Edm.PropertyPath`. It uses the same syntax and rules as the `edm:Path` expression, with the following exceptions:

- The `PropertyPath` expression may traverse multiple collection-valued structural or navigation properties

- The last path segment MUST resolve either to a structural property in the context of the preceding path part, or to a term cast where the term MUST be of type `Edm.ComplexType`, `Edm.PrimitiveType`, a complex type, an enumeration type, a concrete primitive type, a type definition, or a collection of one of these types.

In contrast to the `edm:Path` expression, the value of the `edm:PropertyPath` expression is the path itself, not the value of the property identified by the path.

The `edm:PropertyPath` MAY be provided using either element notation or attribute notation.

Example 68:

```
<Annotation Term="UI.RefreshOnChangeOf" PropertyPath="ChangedAt" />

<Annotation Term="Capabilities.UpdateRestrictions">
  <Record>
    <PropertyValue Property="NonUpdatableProperties">
      <Collection>
        <PropertyPath>CreatedAt</PropertyPath>
        <PropertyPath>ChangedAt</PropertyPath>
      </Collection>
    </PropertyValue>
  </Record>
</Annotation>
```

14.5.14 Expression `edm:Record`

The `edm:Record` expression enables a new entity type or complex type instance to be constructed.

A record expression contains zero or more `edm:PropertyValue` elements. For each single-valued structural or navigation property of the record construct's type that is neither nullable nor specifies a default value an `edm:PropertyValue` child element MUST be provided. The only exception is if the record expression is the direct child of an `edm:Annotation` element for a term that has a base term whose type is structured and directly or indirectly inherits from the type of its base term. In this case, property values that already have been specified in the annotation for the base term or its base term etc. need not be specified again.

For collection-valued properties the absence of an `edm:PropertyValue` child element is equivalent to specifying a child element with an empty collection as its value.

A record expression MUST be written with element notation, as shown in the following example.

Example 69: record with two structural and two navigation properties

```
<Annotation Term="org.example.person.Employee">
  <Record>
    <PropertyValue Property="GivenName" Path="FirstName" />
    <PropertyValue Property="Surname" Path="LastName" />
    <PropertyValue Property="Manager" Path="DirectSupervisor" />
    <PropertyValue Property="CostCenter">
      <UrlRef>
        <Apply Function="odata.fillUriTemplate">
          <String>http://host/anotherservice/CostCenters('{ccid}')</String>
          <LabeledElement Name="ccid" Path="CostCenterID" />
        </Apply>
      </UrlRef>
    </PropertyValue>
  </Record>
</Annotation>
```

14.5.14.1 Attribute `Type`

A record expression MAY specify a QualifiedName value for the `Type` attribute that MUST resolve to an entity type or complex type in scope. If no value is specified for the type attribute, the type is derived from the expression's context.

14.5.14.2 Element `edm:PropertyValue`

The `edm:PropertyValue` element supplies a value to a property on the type instantiated by an `edm:Record` expression. The value is obtained by evaluating an expression.

The `PropertyValue` element MUST contain exactly one expression. The `edm:PropertyValue` expression MAY be provided using element notation or attribute notation.

14.5.14.2.1 Attribute `Property`

The `PropertyValue` element MUST assign a SimpleIdentifier value to the `Property` attribute. The value of the property attribute MUST resolve to a property of the type of the enclosing `edm:Record` expression.

14.5.15 Expression `edm:UrlRef`

The `edm:UrlRef` expression enables a value to be obtained by sending a `GET` request to the value of the `UrlRef` expression.

The `edm:UrlRef` element MUST contain exactly one expression of type `Edm.String`. The `edm:UrlRef` expression MAY be provided using element notation or attribute notation.

The URL may be relative or absolute; relative URIs are relative to the `xml:base` attribute, see **[XML-Base]**.

The response body of the `GET` request MUST be returned as the result of the `edm:UrlRef` expression. The result of the `edm:UrlRef` expression MUST be type compatible with the type expected by the surrounding element or expression.

Example 70:

```
<Annotation Term="Vocab.Supplier">
  <UrlRef>
    <Apply Function="odata.fillUriTemplate">
      <String>http://host/service/Suppliers({suppID})</String>
      <LabeledElement Name="suppID">
      <Apply Function="odata.uriEncode">
        <Path>SupplierId</Path>
      </Apply>
      </LabeledElement>
     </Apply>
  </UrlRef>
</Annotation>

<Annotation Term="Core.LongDescription">
  <UrlRef><String>http://host/wiki/HowToUse</String></UrlRef>
</Annotation>

<Annotation Term="Core.LongDescription" UrlRef="http://host/wiki/HowToUse" />
```

15 Metadata Service Schema

The Metadata Service is a representation of the entity model of an OData service as an OData service with a fixed (meta) data model. The Metadata Service provides convenient access to the entity model of a service, i.e. all CSDL constructs used in its entity containers.

With `~/` as an abbreviation for the service root URL, the Metadata Service root URL is `~/$metadata/`, i.e. the canonical URL of the metadata document of the underlying service with a forward slash appended, and a `GET` request to `~/$metadata/$metadata` returns the CSDL document of the Metadata Service itself, defined in **[OData-Meta]**.

The following sections describe the schema of the Metadata Service.

Example 71: service document of Metadata Service

```
GET ~/$metadata/
```

would return

```
{
  "@odata.context":"~/$metadata/$metadata",
  "value":[
    { "name":"References"                ,"url":"References" },
    { "name":"Schemata"                  ,"url":"Schemata" },
    { "name":"Types"                     ,"url":"Types" },
    { "name":"Properties"                ,"url":"Properties" },
    { "name":"NavigationProperties"      ,"url":"NavigationProperties" },
    { "name":"EnumTypeMembers"           ,"url":"EnumTypeMembers" },
    { "name":"Actions"                   ,"url":"Actions" },
    { "name":"Functions"                 ,"url":"Functions" },
    { "name":"Terms"                     ,"url":"Terms" },
    { "name":"Annotations"               ,"url":"Annotations" },
    { "name":"EntityContainer"           ,"url":"EntityContainer",
                                          "kind":"Singleton" },
    { "name":"EntitySets"                ,"url":"EntitySets" },
    { "name":"Singletons"                ,"url":"Singletons" },
    { "name":"NavigationPropertyBindings","url":"NavigationPropertyBindings"},
    { "name":"ActionImports"             ,"url":"ActionImports" },
    { "name":"FunctionImports"           ,"url":"FunctionImports" }
  ]
}
```

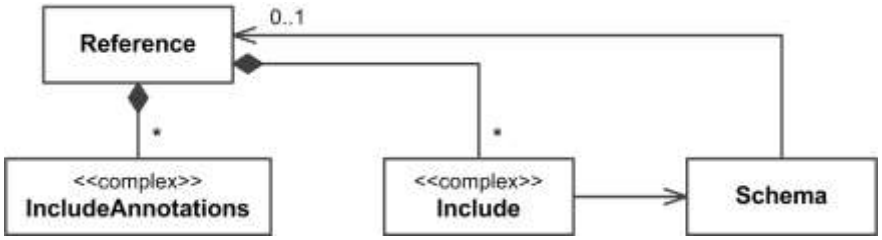
Note: all examples in this chapter use `~/` as an abbreviation for the service root URL.

Note: `~/$metadata/$metadata` is not a typo, it is the metadata URL of the Metadata Service for the service with root URL `~/`.

15.1 Entity Model Wrapper

The Metadata Service provides convenient access to the entity model of a service, i.e. all CSDL constructs used in its entity container. This model may be distributed over several schemas, and these schemas may be distributed over several physical locations, bound together via the entity model wrapper.

This document structure is represented in the metadata service as an entity type `Reference` and two complex types `Include` and `IncludeAnnotations`.



Legend: boxes without a stereotype represent entity types; boxes with stereotype «`complex`» represent complex types. Compositions represent complex properties; associations represent navigation properties. Arrows indicate navigation properties without a partner; associations without arrows are bidirectional. No cardinality means 1.

A reference is identified by its `Uri` property, which is the absolute value of the `Uri` attribute after resolving a relative value against the `xml:base` attribute.

Example 72: for the Products and Categories example the request

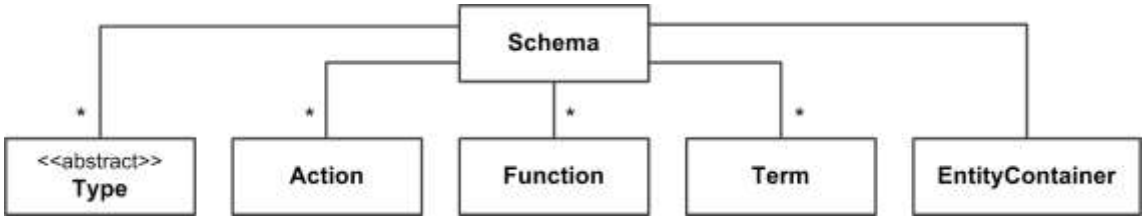
```
GET ~/$metadata/References?$expand=Include/Schema($select=Namespace)
```

would return

```
{
  "@odata.context":
              "~/$metadata/$metadata#References(.,Include/Schema(Namespace))",
  "value":[
    {
      "Uri":"http://tinyurl.com/Org-OData-Core",
      "Include":[
        { "Alias":"Core", "Schema":{ "Namespace":"Org.OData.Core.V1" } }
      ],
      "IncludeAnnotations":[]
    },{
      "Uri":"http://tinyurl.com/Org-OData-Measures-V1",
      "Include":[
        { "Alias":"UoM", "Schema":{ "Namespace":"Org.OData.Measures.V1" } }
      ],
      "IncludeAnnotations":[]
    }
  ]
}
```

15.2 Schema

The *model* of the service consists of all CSDL constructs used in its entity container. Each model construct is defined in a schema:



A schema is identified by its `Namespace` property. If it defines an alias, direct key access using the alias instead of the namespace redirects to the schema with this alias.

Example 73: for the Products and Categories example the request

```
GET ~/$metadata/Schemata
```

would return

```
{
  "@odata.context":"~/$metadata#Schemata",
"value":[
  {
      "Namespace":"ODataDemo", "Alias":null
    },{
      "Namespace":"Org.OData.Core.V1", "Alias":"Core"
    },{
      "Namespace":"Org.OData.Measures.V1", "Alias":"UoM"
    },{
      "Namespace":"Edm", "Alias":null
    }
  ]
}
```

Example 74: redirecting from alias to schema

```
GET ~/$metadata/Schemata('Core')
```

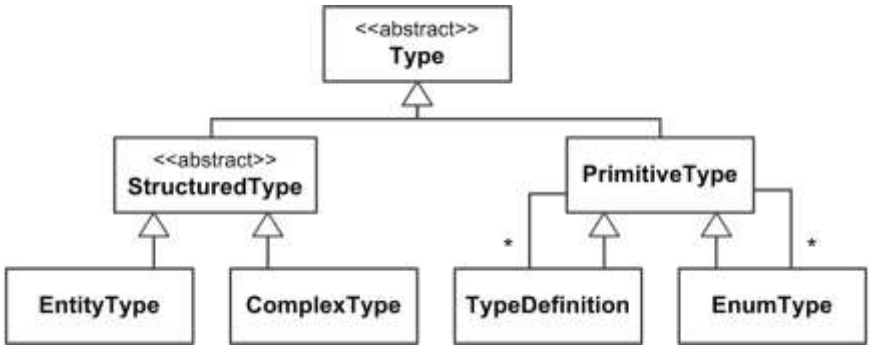
would return

```
{
  "@odata.context":"~/$metadata#Schemata/@entity",
"Namespace":"Org.OData.Core.V1",
"Alias":"Core"
}
```

All schemata used in the model are listed in this entity set, independently of whether they are defined directly in the metadata document or included via a reference.

15.3 Types

Types form an inheritance hierarchy



A type is identified by its `QualifiedName` property, which is the `Namespace` of the defining schema, followed by a dot (`.`) and the `Name` of the type. There is only one entity set `Types` for all types. Type cast segments can be used to access specialized types.

Only those built-in primitive types that are actually used in the model appear in the `Types` entity set.

Enumeration type members are identified by their `Fullname` property, which is the `QualifiedName` of the enumeration type, followed by a forward slash (`/`) and the `Name` of the member.

Example 75: single type by name, and all entity types

```
GET ~/$metadata/Types('ODataDemo.Product')
GET ~/$metadata/Types/Meta.EntityType
```

Example 76: all types

```
GET ~/$metadata/Types
```

would return

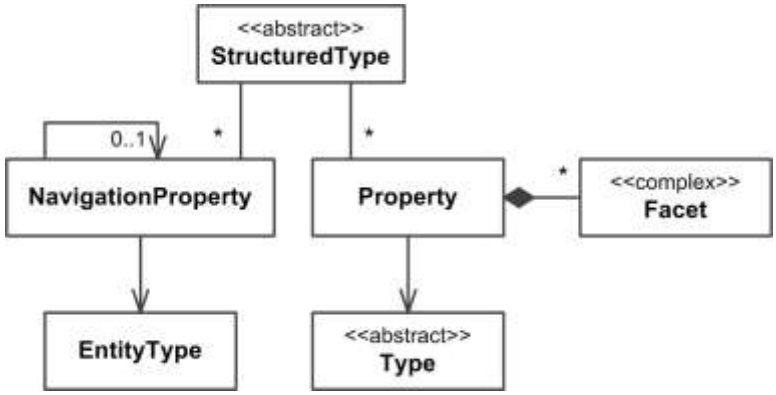
```
{
  "@odata.context":"~/$metadata/$metadata#Types",
  "value":[ {
      "@odata.type":"Meta.EntityType",
      "QualifiedName":"ODataDemo.Product", "Name":"Product",
      "Key":[{"PropertyPath":"ID","Alias":null}],
      "Abstract":false, "OpenType":false, "HasStream":true
    },{
      "@odata.type":"Meta.EntityType",
      "QualifiedName":"ODataDemo.Category", "Name":"Category",
      "Key":[{"PropertyPath":"ID","Alias":null}],
      "Abstract":false, "OpenType":false, "HasStream":false
    },{
      "@odata.type":"Meta.EntityType",
      "QualifiedName":"ODataDemo.Supplier", "Name":"Supplier",
      "Key":[{"PropertyPath":"ID","Alias":null}],
      "Abstract":false, "OpenType":false, "HasStream":false
    },{
      "@odata.type":"Meta.EntityType",
      "QualifiedName":"ODataDemo.Country", "Name":"Country",
      "Key":[{"PropertyPath":"Code","Alias":null}],
      "Abstract":false, "OpenType":false, "HasStream":false
    },{
      "@odata.type":"Meta.ComplexType",
      "QualifiedName":"ODataDemo.Address", "Name":"Address",
      "Abstract":false, "OpenType":false
    },{
```



```
      "@odata.type":"Meta.ComplexType",
      "QualifiedName":"Core.OptimisticConcurrency",
      "Name":"OptimisticConcurrency",
      "Abstract":false, "OpenType":false
    },{
      "@odata.type":"Meta.PrimitiveType",
      "QualifiedName":"Edm.Date", "Name":"Date"
    },{
      "@odata.type":"Meta.PrimitiveType",
      "QualifiedName":"Edm.Decimal", "Name":"Decimal"
    },{
      "@odata.type":"Meta.PrimitiveType",
      "QualifiedName":"Edm.Int32", "Name":"Int32"
    },{
      "@odata.type":"Meta.PrimitiveType",
      "QualifiedName":"Edm.String", "Name":"String"
    },{
      "@odata.type":"Meta.PrimitiveType",
      "QualifiedName":"Edm.PropertyPath", "Name":"PropertyPath"
    },{
      "@odata.type":"Meta.EntityType",
      "QualifiedName":"Edm.EntityType", "Name":"EntityType", "Key":[],
      "Abstract":true, "OpenType":false, "HasStream":false    }
  ]
}
```

15.4 Properties

Structural properties and navigation properties are represented as



This model is intentionally simplified. It closely resembles the XML schema and makes querying easy as it e.g. allows expanding the `Type` for all structural properties. A structured type is only related to properties it directly declares, not to properties it inherits from ancestor types. All inherited and directly declared properties or navigation properties can be requested with the bound functions `Meta.AllProperties` and `Meta.AllNavigationProperties`.

Structural properties and navigation properties are identified by their `Fullname` property, which is the `QualifiedName` of the containing entity type or complex type, followed by a forward slash (`/`) and the `Name` of the property or navigation property.

Example 77: single property or navigation property by name

```
GET ~/$metadata/Properties('ODataDemo.Product%2FID')
GET ~/$metadata/NavigationProperties('ODataDemo.Category%2FProducts')
```

Example 78: all properties with type

```
GET ~/$metadata/Properties?$expand=Type($select=QualifiedName)
```

would return

```
{
  "@odata.context":"~/$metadata/$metadata#Properties(*,Type(QualifiedName))",
  "value":[
    {
      "Fullname":"ODataDemo.Product/ID", "Name":"ID",
      "Nullable":false, "IsCollection":false,
      "Type":{"QualifiedName":"Edm.String"},
      "Facets":[]
    },{
      "Fullname":"ODataDemo.Product/Description", "Name":"Description",
      "Nullable":false, "IsCollection":false,
      "Type":{"QualifiedName":"Edm.String"},
      "Facets":[]
    },{
      "Fullname":"ODataDemo.Product/ReleaseDate", "Name":"ReleaseDate",
      "Nullable":true, "IsCollection":false,
      "Type":{"QualifiedName":"Edm.Date"},
      "Facets":[]
    },{
      "Fullname":"ODataDemo.Product/DiscontinuedDate",
      "Name":"DiscontinuedDate",
      "Nullable":true, "IsCollection":false,
      "Type":{"QualifiedName":"Edm.Date"},
      "Facets":[]
    },{
      "Fullname":"ODataDemo.Product/Rating", "Name":"Rating",
      "Nullable":true, "IsCollection":false,
      "Type":{"QualifiedName":"Edm.Int32"},
      "Facets":[]
    },{
      "Fullname":"ODataDemo.Product/Currency", "Name":"Currency",
      "Nullable":false, "IsCollection":false,
      "Type":{"QualifiedName":"Edm.String"},
      "Facets":[{"Name":"MaxLength","Value":"3"}]
    },{
      "Fullname":"ODataDemo.Category/ID", "Name":"ID",
      "Nullable":false, "IsCollection":false,
      "Type":{"QualifiedName":"Edm.Int32"},
      "Facets":[]
    },{
      "Fullname":"ODataDemo.Category/Name", "Name":"Name",
      "Nullable":false, "IsCollection":false,
      "Type":{"QualifiedName":"Edm.String"},
      "Facets":[]
    },{
      "Fullname":"ODataDemo.Supplier/ID", "Name":"ID",
      "Nullable":false, "IsCollection":false,
      "Type":{"QualifiedName":"Edm.String"},
      "Facets":[]
    },{
      "Fullname":"ODataDemo.Supplier/Name", "Name":"Name",
      "Nullable":false, "IsCollection":false,
      "Type":{"QualifiedName":"Edm.String"},
      "Facets":[]
    },{
      "Fullname":"ODataDemo.Supplier/Address", "Name":"Address",
      "Nullable":false, "IsCollection":false,
      "Type":{"QualifiedName":"ODataDemo.Address"},
      "Facets":[]
    },{
```



```
      "@odata.type":"Meta.PrimitiveProperty",
      "Fullname":"ODataDemo.Supplier/Concurrency", "Name":"Concurrency",
      "Nullable":false, "IsCollection":false,
      "Type":{"QualifiedName":"Edm.Int32"},
      "Facets":[]
    },{
      "Fullname":"ODataDemo.Country/Code", "Name":"Code",
      "Nullable":false, "IsCollection":false,
      "Type":{"QualifiedName":"Edm.String"},
      "Facets":[{"Name":"MaxLength","Value":"2"}]
    },{
      "Fullname":"ODataDemo.Country/Name", "Name":"Name",
      "Nullable":false, "IsCollection":false,
      "Type":{"QualifiedName":"Edm.String"},
      "Facets":[]
    },{
      "Fullname":"ODataDemo.Address/Street", "Name":"Street",
      "Nullable":false, "IsCollection":false,
      "Type":{"QualifiedName":"Edm.String"},
      "Facets":[]
    },{
      "Fullname":"ODataDemo.Address/City", "Name":"City",
      "Nullable":false, "IsCollection":false,
      "Type":{"QualifiedName":"Edm.String"},
      "Facets":[]
    },{
      "Fullname":"ODataDemo.Address/State", "Name":"State",
      "Nullable":false, "IsCollection":false,
      "Type":{"QualifiedName":"Edm.String"},
      "Facets":[]
    },{
      "Fullname":"ODataDemo.Address/ZipCode", "Name":"ZipCode",
      "Nullable":false, "IsCollection":false,
      "Type":{"QualifiedName":"Edm.String"},
      "Facets":[]
    },{
      "Fullname":"ODataDemo.Address/CountryName", "Name":"CountryName",
      "Nullable":false, "IsCollection":false,
      "Type":{"QualifiedName":"Edm.String"},
      "Facets":[]
    },{
      "Fullname":"Core.OptimisticConcurrency/ETagDependsOn",
      "Name":"ETagDependsOn",
      "Nullable":false, "IsCollection":true,
      "Type":{"QualifiedName":"Edm.PropertyPath"},
      "Facets":[]
    }
  ]
}
```

Example 79: all navigation properties with type and partner

```
GET ~/$metadata/NavigationProperties?
                    $expand=Type($select=QualifiedName),Partner($select=Name)
```

would return

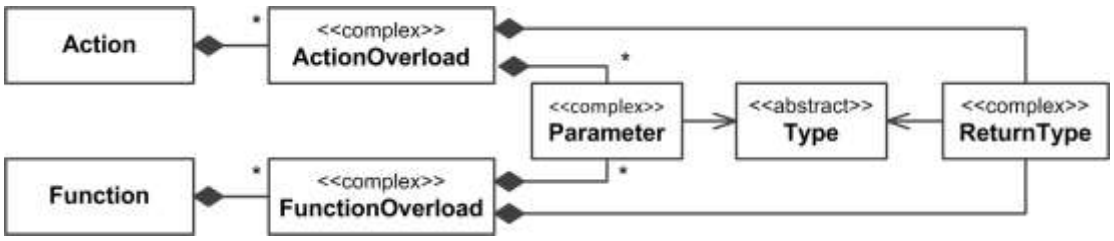
```
{
"@odata.context":"~/$metadata/$metadata#NavigationProperties(Type(QualifiedNam
e),Partner(Name))",
    {
      "Fullname":"ODataDemo.Product/Category", "Name":"Category",
      "Nullable":false, "ContainsTarget":false,
      "OnDelete":null,
```



```
      "ReferentialConstraints":[],
      "IsCollection":false,
      "Type":{ "QualifiedName":"ODataDemo.Category" },
      "Partner":{ "Name":"Product" }
    },{
      "Fullname":"ODataDemo.Product/Supplier", "Name":"Supplier",
      "Nullable":false, "ContainsTarget":false,
      "OnDelete":null,
      "ReferentialConstraints":[],
      "IsCollection":false,
      "Type":{ "QualifiedName":"ODataDemo.Supplier" },
      "Partner":{ "Name":"Products" }
    },{
      "Fullname":"ODataDemo.Category/Products", "Name":"Products",
      "Nullable":false, "ContainsTarget":false,
      "OnDelete":{ "Action":"Cascade", "Annotations":[] },
      "ReferentialConstraints":[],
      "IsCollection":true,
      "Type":{ "QualifiedName":"ODataDemo.Product" },
      "Partner":{ "Name":"Category" }
    },{
      "Fullname":"ODataDemo.Supplier/Products", "Name":"Products",
      "Nullable":false, "ContainsTarget":false,
      "OnDelete":null,
      "ReferentialConstraints":[],
      "IsCollection":true,
      "Type":{ "QualifiedName":"ODataDemo.Product" },
      "Partner":{ "Name":"Supplier" }
    },{
      "Fullname":"ODataDemo.Address/Country", "Name":"Country",
      "Nullable":false, "ContainsTarget":false,
      "OnDelete":null,
      "ReferentialConstraints":[
        {
          "Property":"CountryName", "ReferencedProperty":"Name",
          "Annotations":[]
        }
      ],
      "IsCollection":false,
      "Type":{ "QualifiedName":"ODataDemo.Product" },
      "Partner":{ "Name":"Supplier" }
    }
  ]
}
```

15.5 Actions and Functions

Actions and functions are represented as



Actions and functions are identified by their `QualifiedName` property, which is the `Namespace` of the containing schema, followed by a dot (`.`) and the `Name` of the action or function.

Example 80:

```
GET ~/$metadata/Actions('SampleModel.Approval')
GET ~/$metadata/Functions('ODataDemo.ProductsByRating')
```

Example 81: all functions

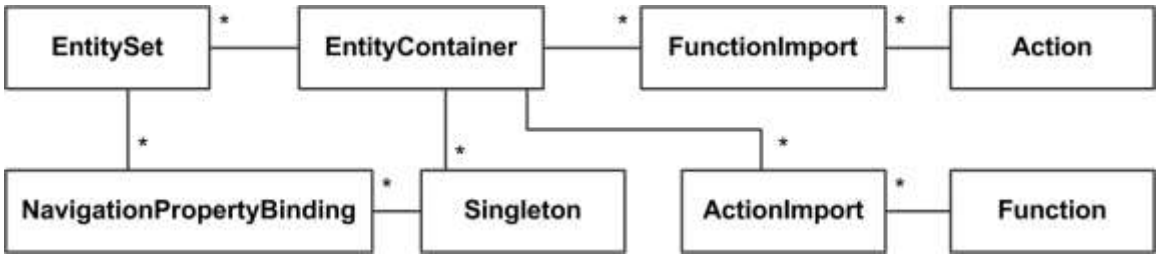
```
GET ~/$metadata/Functions?
                    $expand=Overloads/Parameters/Type($select=QualifiedName)
```

would return

```
{
  "@odata.context":
"~/$metadata/$metadata#Functions(*,Overloads/Parameters/Type(QualifiedName))",
  "value":[
    {
      "QualifiedName":"ODataDemo.ProductsByRating",
      "Name":"ProductsByRating",
      "Overloads":[
        {
          "IsBound":false, "IsComposable":false,
          "ReturnType":{
            "IsCollection":true, "Nullable":false, "Facets":[],
            "Type":{"QualifiedName":"ODataDemo.Product"}
          },
          "Parameters":[
            {
              "Name":"Rating", "IsBinding":false,
              "Nullable":true,
              "IsCollection":false, "Facets":[],
              "Type":{"QualifiedName":"Edm.Int32"}
            }
          ]
        }
      ]
    }
  ]
}
```

15.6 Entity Container

Entity container constructs are represented as



An entity container is identified by its `QualifiedName` property, which is the `Namespace` of the containing schema, followed by a dot (`.`) and the `Name` of the entity container. As there is exactly one entity container per service, it is a singleton.

Example 82:

```
GET ~/$metadata/EntityContainer
```

Direct children of an entity container are identified by their `Fullname` property, which is the `QualifiedName` of the entity container, followed by a forward slash (`/`) and the `Name` of the child.

Example 83:

```
GET ~/$metadata/EntitySets('ODataDemo.DemoService%2FCategories')
```

A navigation property binding is identified by its `Fullname` property, which is the `Fullname` of the source entity set or singleton, followed by a forward slash (`/`) and the `Path` of the navigation property binding.

Example 84:

```
GET ~/$metadata/NavigationPropertyBindings(
                              'ODataDemo.DemoService%2FCategories%2FProducts')
```

Example 85: all containers with direct children

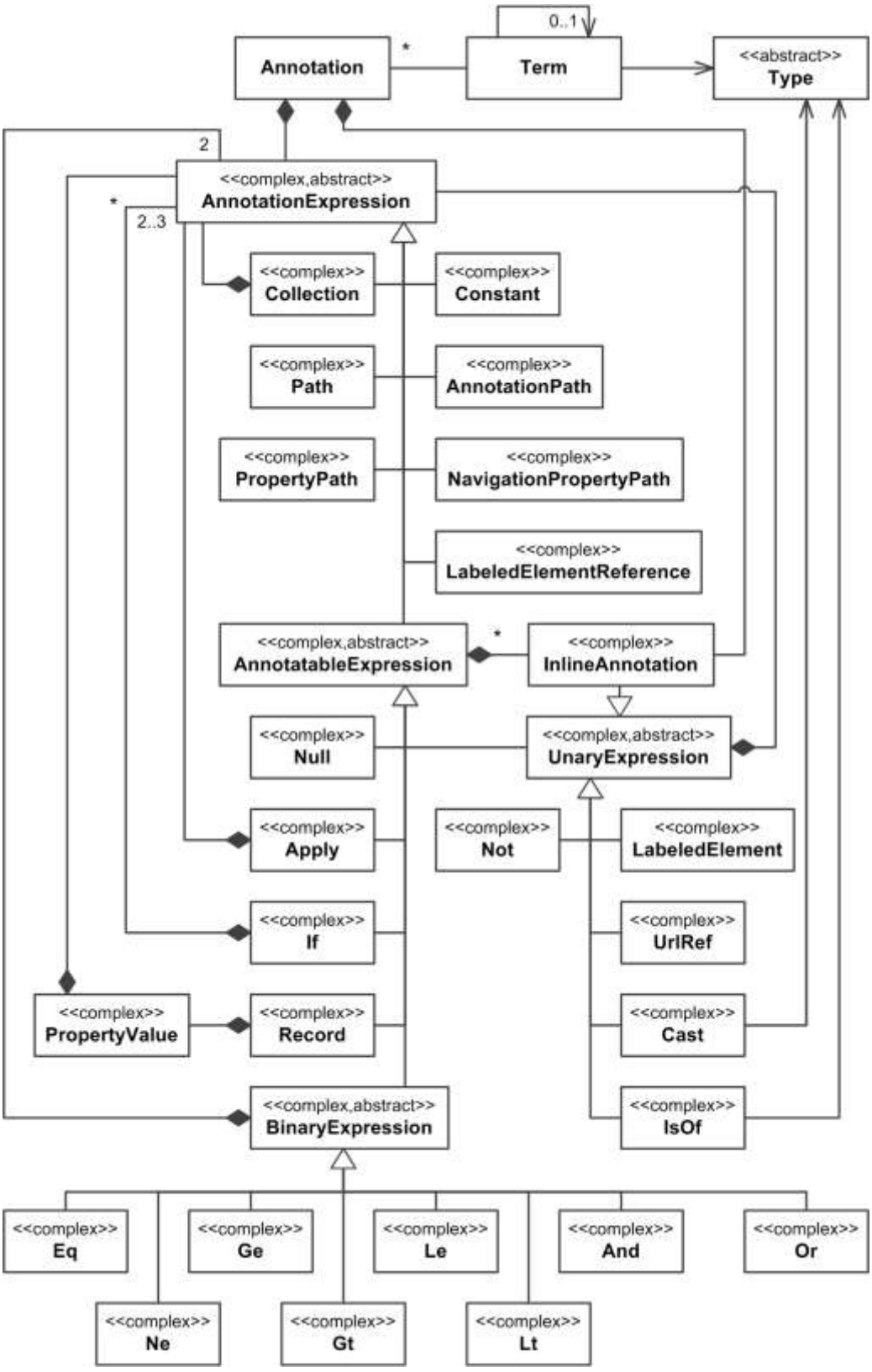
```
GET ~/$metadata/EntityContainer?$expand=*
```

would return

```
{
  "@odata.context":"~/$metadata/$metadata#EntityContainer",
  "value":[
    {
      "QualifiedName":"ODataDemo.DemoService",
      "Name":"DemoService",
      "EntitySets":[
        {
          "Fullname":"ODataDemo.DemoService/Products",   "Name":"Products",
          "IncludeInServiceDocument":true
        },{
          "Fullname":"ODataDemo.DemoService/Suppliers",  "Name":"Suppliers",
          "IncludeInServiceDocument":true
        },{
          "Fullname":"ODataDemo.DemoService/Categories", "Name":"Categories",
          "IncludeInServiceDocument":true
        },{
          "Fullname":"ODataDemo.DemoService/Countries",  "Name":"Countries",
          "IncludeInServiceDocument":true
        }
      ],
      "Singletons":[
        {
          "QualifiedName":"ODataDemo.DemoService/Contoso", "Name":"Contoso"
        }],
      "ActionImports":[],
      "FunctionImports":[
        {
          "QualifiedName":"ODataDemo.DemoService/ProductsByRating",
          "Name":"ProductsByRating",
          "IncludeInServiceDocument":false
        }
      ]
    }
  ]
}
```


15.7 Terms and Annotations

Terms and annotations based on these terms are represented as



A term is identified by its `QualifiedName` property, which is the `Namespace` of the containing schema, followed by a dot (`.`) and the `Name` of the term.

Example 86:

```
GET ~/$metadata/Terms?$expand=Type($select=QualifiedName)
```

would return

```
{
  "@odata.context":"~/$metadata/$metadata#Terms(Type(QualifiedName))",
  "value":[
    {
      "QualifiedName":"Core.Description", "Name":"Description",
      "DefaultValue":null, "IsCollection":false,
      "Type":{ "QualifiedName":"Edm.String" }
    },{
      "QualifiedName":"Core.OptimisticConcurrency",
      "Name":"OptimisticConcurrency",
      "DefaultValue":null, "IsCollection":true,
      "Type":{ "QualifiedName":"Edm.PropertyPath" }
    }
]
}
```

Annotations can be stated in CSDL in two ways: *inline* as child elements of the annotated element, or *externally* as children of an `edm:Annotations` element that targets the model element to be annotated. The external form is only possible for model elements that can be uniquely identified by a target path expression, and these model elements are represented in the Metadata Service as entity types, while all model elements that cannot be targeted are represented as complex types.

Consequently annotations that can only be stated with the inline form are represented with the complex type `Edm.Metadata.InlineAnnotation`, while annotations that can be stated externally are represented with the entity type `Edm.Metadata.Annotation`, whether they are stated inline or externally in the metadata document or referenced CSDL documents. If the example metadata document in Example 88 would reference the CSDL document in Example 89, all its annotations would also be members of the Annotations entity set of the Metadata Service for Example 88.

These annotations are identified by the combination of their target, term, and qualifier. The `Fullname` of an annotation is the `Fullname` of the target, followed by an at (`@`) sign and the `QualifiedName` of the term, and for non-empty qualifiers followed by a hash (`#`) sign and the qualifier.

Example 87:

```
GET ~/$metadata/Annotations
```

would return

```
{
  "@odata.context":"~/$metadata/$metadata#Annotations",
  "value":[
    {
      "Fullname":"ODataDemo.Product/Description@Core.IsLanguageDependent",
      "Qualifier":null,
      "Value":{ "@odata.type":"Meta.ConstantExpression","Value":true }
    },
    {
      "Fullname":"ODataDemo.Product/Price@UoM.ISOCurrency",
      "Qualifier":null,
      "Value":{ "@odata.type":"Meta.Path","Value":"Currency" }
    },
    {
      "Fullname":"ODataDemo.Category/Name@Core.IsLanguageDependent",
      "Qualifier":null,
```



```
      "Value":{ "@odata.type":"Meta.Constant","Value":true }
    },
    {
      "Fullname":"ODataDemo.DemoService/Suppliers@Core.OptimisticConcurrency",
      "Qualifier":null,
      "Value":{
        "@odata.type":"Meta.Collection",
        "Items":[
          {
            "@odata.type":"Meta.PropertyPath",
            "Value":"Concurrency"
          }
        ]
      }
    }
  ]
}
```

16 CSDL Examples

Following are two basic examples of valid EDM models as represented in CSDL. These examples demonstrate many of the topics covered above.

16.1 Products and Categories Example

Example 88:

```
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx"
           Version="4.0">
  <edmx:Reference Uri="http://docs.oasis-
open.org/odata/odata/v4.0/os/vocabularies/Org.OData.Core.V1.xml">
    <edmx:Include Namespace="Org.OData.Core.V1" Alias="Core" />
</edmx:Reference>
  <edmx:Reference Uri="http://docs.oasis-
open.org/odata/odata/v4.0/os/vocabularies/Org.OData.Measures.V1.xml">
    <edmx:Include Alias="UoM" Namespace="Org.OData.Measures.V1" />
</edmx:Reference>
  <edmx:DataServices>
    <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm"
            Namespace="ODataDemo">
      <EntityType Name="Product" HasStream="true">
        <Key>
          <PropertyRef Name="ID" />
        </Key>
        <Property Name="ID" Type="Edm.Int32" Nullable="false" />
        <Property Name="Description" Type="Edm.String" >
          <Annotation Term="Core.IsLanguageDependent" />
        </Property>
        <Property Name="ReleaseDate" Type="Edm.Date" />
        <Property Name="DiscontinuedDate" Type="Edm.Date" />
        <Property Name="Rating" Type="Edm.Int32" />
        <Property Name="Price" Type="Edm.Decimal">
          <Annotation Term="UoM.ISOCurrency" Path="Currency" />
        </Property>
        <Property Name="Currency" Type="Edm.String" MaxLength="3" />
        <NavigationProperty Name="Category" Type="ODataDemo.Category"
                            Nullable="false" Partner="Products" />
        <NavigationProperty Name="Supplier" Type="ODataDemo.Supplier"
                            Partner="Products" />
      </EntityType>
      <EntityType Name="Category">
        <Key>
         <PropertyRef Name="ID" />
        </Key>
        <Property Name="ID" Type="Edm.Int32" Nullable="false" />
        <Property Name="Name" Type="Edm.String">
          <Annotation Term="Core.IsLanguageDependent" />
        </Property>
        <NavigationProperty Name="Products" Partner="Category"
                            Type="Collection(ODataDemo.Product)">
          <OnDelete Action="Cascade" />
```

```
      </NavigationProperty>
    </EntityType>
    <EntityType Name="Supplier">
      <Key>
        <PropertyRef Name="ID" />
      </Key>
      <Property Name="ID" Type="Edm.String" Nullable="false" />
      <Property Name="Name" Type="Edm.String" />
      <Property Name="Address" Type="ODataDemo.Address" Nullable="false" />
      <Property Name="Concurrency" Type="Edm.Int32" Nullable="false" />
      <NavigationProperty Name="Products" Partner="Supplier"
                          Type="Collection(ODataDemo.Product)" />
    </EntityType>
    <EntityType Name="Country">
      <Key>
        <PropertyRef Name="Code" />
      </Key>
      <Property Name="Code" Type="Edm.String" MaxLength="2"
                            Nullable="false" />
      <Property Name="Name" Type="Edm.String" />
    </EntityType>
    <ComplexType Name="Address">
      <Property Name="Street" Type="Edm.String" />
      <Property Name="City" Type="Edm.String" />
      <Property Name="State" Type="Edm.String" />
      <Property Name="ZipCode" Type="Edm.String" />
      <Property Name="CountryName" Type="Edm.String" />
      <NavigationProperty Name="Country" Type="ODataDemo.Country">
        <ReferentialConstraint Property="CountryName"
                               ReferencedProperty="Name" />
      </NavigationProperty>
    </ComplexType>
    <Function Name="ProductsByRating">
      <Parameter Name="Rating" Type="Edm.Int32" />
      <ReturnType Type="Collection(ODataDemo.Product)" />
    </Function>
    <EntityContainer Name="DemoService">
      <EntitySet Name="Products" EntityType="ODataDemo.Product">
        <NavigationPropertyBinding Path="Category" Target="Categories" />
      </EntitySet>
      <EntitySet Name="Categories" EntityType="ODataDemo.Category">
        <NavigationPropertyBinding Path="Products" Target="Products" />
      </EntitySet>
      <EntitySet Name="Suppliers" EntityType="ODataDemo.Supplier">
        <NavigationPropertyBinding Path="Products" Target="Products" />
        <NavigationPropertyBinding Path="Address/Country"
                                   Target="Countries" />
        <Annotation Term="Core.OptimisticConcurrency">
          <Collection>
            <PropertyPath>Concurrency</PropertyPath>
          </Collection>
        </Annotation>
      </EntitySet>
      <Singleton Name="Contoso" Type="Self.Supplier">
        <NavigationPropertyBinding Path="Products" Target="Products" />
      </Singleton>
      <EntitySet Name="Countries" EntityType="ODataDemo.Country" />
      <FunctionImport Name="ProductsByRating" EntitySet="Products"
                      Function="ODataDemo.ProductsByRating" />
    </EntityContainer>
  </Schema>
 </edmx:DataServices>
</edmx:Edmx>
```


16.2 Annotations for Products and Categories Example

Example 89:

```
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx"
           Version="4.0">
  <edmx:Reference Uri="http://host/service/$metadata">
    <edmx:Include Namespace="ODataDemo" />
  </edmx:Reference>
  <edmx:Reference Uri="http://somewhere/Vocabulary/V1">
    <edmx:Include Alias="Vocabulary1" Namespace="Some.Vocabulary.V1" />
</edmx:Reference>
<edmx:DataServices>
    <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm"
            Namespace="Annotations">
      <Annotations Target="ODataDemo.Supplier">
        <Annotation Term="Vocabulary1.EMail">
          <Null />
        </Annotation>
        <Annotation Term="Vocabulary1.AccountID" Path="ID" />
        <Annotation Term="Vocabulary1.Title" String="Supplier Info" />
        <Annotation Term="Vocabulary1.DisplayName">
        <Apply Function="odata.concat">
            <Path>Name</Path>
            <String> in </String>
            <Path>Address/CountryName</Path>
          </Apply>
        </Annotation>
      </Annotations>
      <Annotations Target="ODataDemo.Product">
        <Annotation Term="Vocabulary1.Tags">
          <Collection>
            <String>MasterData</String>
          </Collection>
        </Annotation>
      </Annotations>
  </Schema>
  </edmx:DataServices>
</edmx:Edmx>
```


17 Attribute Values

17.1 Namespace

A Namespace is a character sequence of type `edm:TNamespaceName`, see **[OData-EDM]**.

Non-normatively speaking it is a dot-separated sequence of SimpleIdentifiers with a maximum length of 511 Unicode characters.

17.2 SimpleIdentifier

A SimpleIdentifier is a character sequence of type `edm:TSimpleIdentifier`, see **[OData-EDM]**:

```
<xs:simpleType name="TSimpleIdentifier">
  <xs:restriction base="xs:NCName">
    <xs:maxLength value="128" />
    <xs:pattern
     value="[\p{L}\p{Nl}_][\p{L}\p{Nl}\p{Nd}\p{Mn}\p{Mc}\p{Pc}\p{Cf}]{0,}"
    />
  </xs:restriction>
</xs:simpleType>
```

Non-normatively speaking it starts with a letter or underscore, followed by at most 127 letters, underscores or digits.

17.3 QualifiedName

For model elements that are direct children of a schema: the namespace or alias of the schema that defines the model element, followed by a dot and the name of the model element, see rule `qualifiedTypeName` in **[OData-ABNF]**.

For built-in primitive types: the name of the type, prefixed with `Edm` followed by a dot.

17.4 TypeName

The QualifiedName of a built-in primitive or abstract type, a type definition, complex type, enumeration type, or entity type, or a collection of one of these types, see rule `qualifiedTypeName` in **[OData-ABNF]**.

The type must be in scope, i.e. the type MUST be defined in the `Edm` namespace or it MUST be defined in the schema identified by the namespace or alias portion of the qualified name, and the identified schema MUST be defined in the same CSDL document or included from a directly referenced document.

17.5 TargetPath

Target paths are used in attributes of CSDL elements to refer to other CSDL elements or their nested child elements.

The allowed path expressions are:

- The QualifiedName of an entity container, followed by a forward slash and the name of a container child element
- The target path of a container child followed by a forward slash and one or more forward-slash separated property, navigation property, or type cast segments

Example 90: Target expressions

```
MySchema.MyEntityContainer/MyEntitySet
```



```
MySchema.MyEntityContainer/MySingleton
MySchema.MyEntityContainer/MyEntitySet/MyContainmentNavigationProperty
MySchema.MyEntityContainer/MyEntitySet/My.EntityType/MyContainmentNavProperty
MySchema.MyEntityContainer/MySingleton/MyComplexProperty/MyContainmentNavProp
```

17.6 Boolean

One of the literals `true` or `false`.

18 Conformance

Conforming services MUST follow all rules of this specification document for the types, sets, functions, actions, containers and annotations they expose.

Conforming clients MUST be prepared to consume a model that uses any or all of the constructs defined in this specification, including custom annotations, and MUST ignore any elements or attributes not defined in this version of the specification.

Appendix A. Acknowledgments

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

Appendix B. Revision History

Revision	Date	Editor	Changes Made
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	Simplified annotations, relationships, added containment, singletons Added Type Definitions, Edm.Date, Edm.TimeOfDay, Edm.Duration datatypes. Retired Edm.DateTime, Edm.Time. Enhanced ComplexType support Expanded Service Document 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	Restricted services to exactly one entity container Simplified function and action overloads Rounded off annotaitons Fleshed out containment Simplified rules for implicit enum member values Clarified intention of Partner and NavigationPropertyBinding Simplified and completed CSDL for Metadata Service, added description of behavior
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	Changed function overload resolution rules Improved path expressions for annotations
Committee Specification 02	2013-11-04	Michael Pizzo, Ralf Handl, Martin Zurmuehl	Non-Material Changes
OASIS Specification	2014-02-24	Michael Pizzo, Ralf Handl, Martin Zurmuehl	Non-Material Changes
Errata 01	2014-07-24	Michael Pizzo, Ralf Handl, Martin Zurmuehl	Minor changes and improvements
Errata 02	2014-10-29	Michael Pizzo, Ralf Handl,	Repaired mechanical error in the editable

		Martin Zurmuehl	source

ICS 35.100.70

Price based on 203 pages