



OData Version 4.0. Part 3: Common Schema Definition Language (CSDL) Plus Errata 03 (redlined)

OASIS Standard incorporating Approved Errata 03

02 June 2016

Specification URIs

This version:

<http://docs.oasis-open.org/odata/odata/v4.0/errata03/os/redlined/part3-csdl/odata-v4.0-errata03-os-part3-csdl-redlined.doc> (Authoritative)
<http://docs.oasis-open.org/odata/odata/v4.0/errata03/os/redlined/part3-csdl/odata-v4.0-errata03-os-part3-csdl-redlined.html>
<http://docs.oasis-open.org/odata/odata/v4.0/errata03/os/redlined/part3-csdl/odata-v4.0-errata03-os-part3-csdl-redlined.pdf>

Previous 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>

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 SE
Ram Jeyaraman (Ram.Jeyaraman@microsoft.com), Microsoft

Editors:

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

Additional artifacts:

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

- List of Errata items. *OData Version 4.0 Errata 03*. Edited by Michael Pizzo, Ralf Handl, Martin Zurmuehl, and Hubert Heijkers. 02 June 2016. OASIS Approved Errata. <http://docs.oasis-open.org/odata/odata/v4.0/errata03/os/odata-v4.0-errata03-os.html>.
- *OData Version 4.0. Part 1: Protocol Plus Errata 03*. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. 02 June 2016. OASIS Standard incorporating Approved Errata 03.

<http://docs.oasis-open.org/odata/odata/v4.0/errata03/os/complete/part1-protocol/odata-v4.0-errata03-os-part1-protocol-complete.html>.

- *OData Version 4.0. Part 2: URL Conventions Plus Errata 03*. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. 02 June 2016. OASIS Standard incorporating Approved Errata 03. <http://docs.oasis-open.org/odata/odata/v4.0/errata03/os/complete/part2-url-conventions/odata-v4.0-errata03-os-part2-url-conventions-complete.html>.
- *OData Version 4.0. Part 3: Common Schema Definition Language (CSDL) Plus Errata 03*. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. 02 June 2016. OASIS Standard incorporating Approved Errata 03. <http://docs.oasis-open.org/odata/odata/v4.0/errata03/os/complete/part3-csdl/odata-v4.0-errata03-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/errata03/os/complete/abnf/>.
- Vocabulary components: OData Core Vocabulary, OData Measures Vocabulary and OData Capabilities Vocabulary. <http://docs.oasis-open.org/odata/odata/v4.0/errata03/os/complete/vocabularies/>.
- XML schemas: OData EDMX XML Schema and OData EDM XML Schema. <http://docs.oasis-open.org/odata/odata/v4.0/errata03/os/complete/schemas/>.
- OData Metadata Service Entity Model: <http://docs.oasis-open.org/odata/odata/v4.0/errata03/os/complete/models/>.
- Change-marked (redlined) versions of OData Version 4.0 Part 1, Part 2, and Part 3 (this document). OASIS Standard incorporating Approved Errata 03. <http://docs.oasis-open.org/odata/odata/v4.0/errata03/os/redlined/>.

Related work:

This specification is related to:

- *OData Version 4.0 Part 3: Part 3: Common Schema Definition Language (CSDL)*. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. 24 February 2014. OASIS Standard. <http://docs.oasis-open.org/odata/odata/v4.0/os/part3-csdl/odata-v4.0-os-part3-csdl.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 TC's web page (<https://www.oasis-open.org/committees/odata/ipr.php>).

Citation format:

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

[OData-Part3]

OData Version 4.0 Part 3: Common Schema Definition Language (CSDL) Plus Errata 03 (redlined). Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. 02 June 2016. OASIS Standard incorporating Approved Errata 03. <http://docs.oasis-open.org/odata/odata/v4.0/errata03/os/redlined/part3-csdl/odata-v4.0-errata03-os-part3-csdl-redlined.html>. Latest version: <http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part3-csdl.html>.

Notices

Copyright © OASIS Open 2016. All Rights Reserved.

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

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

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

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

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

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

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

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

Table of Contents

1	Introduction.....	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 <code>edm:Edmx</code>	14
3.1.1	Attribute <code>Version</code>	14
3.2	Element <code>edm:DataServices</code>	14
3.3	Element <code>edm:Reference</code>	14
3.3.1	Attribute <code>Uri</code>	15
3.4	Element <code>edm:Include</code>	15
3.4.1	Attribute <code>Namespace</code>	15
3.4.2	Attribute <code>Alias</code>	15
3.5	Element <code>edm:IncludeAnnotations</code>	16
3.5.1	Attribute <code>TermNamespace</code>	16
3.5.2	Attribute <code>Qualifier</code>	16
3.5.3	Attribute <code>TargetNamespace</code>	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 <code>edm:Schema</code>	21
5.1.1	Attribute <code>Namespace</code>	21
5.1.2	Attribute <code>Alias</code>	21
6	Structural Property.....	22
6.1	Element <code>edm:Property</code>	22
6.1.1	Attribute <code>Name</code>	22
6.1.2	Attribute <code>Type</code>	22
6.2	Property Facets.....	22
6.2.1	Attribute <code>Nullable</code>	22
6.2.2	Attribute <code>MaxLength</code>	23
6.2.3	Attribute <code>Precision</code>	23
6.2.4	Attribute <code>Scale</code>	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
12.1.2	Attribute IsBound	38
12.1.3	Attribute EntitySetPath	38
12.2	Element edm:Function	39
12.2.1	Attribute Name	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 <code>edm:Term</code>	48
14.1.1	Attribute <code>Name</code>	48
14.1.2	Attribute <code>Type</code>	48
14.1.3	Attribute <code>BaseTerm</code>	48
14.1.4	Attribute <code>DefaultValue</code>	48
14.1.5	Attribute <code>AppliesTo</code>	48
14.1.6	Term Facets	48
14.2	Element <code>edm:Annotations</code>	49
14.2.1	Attribute <code>Target</code>	49
14.2.2	Attribute <code>Qualifier</code>	50
14.3	Element <code>edm:Annotation</code>	50
14.3.1	Attribute <code>Term</code>	51
14.3.2	Attribute <code>Qualifier</code>	51
14.4	Constant Expressions	51
14.4.1	Expression <code>edm:Binary</code>	51
14.4.2	Expression <code>edm:Bool</code>	52
14.4.3	Expression <code>edm:Date</code>	52
14.4.4	Expression <code>edm:DateTimeOffset</code>	52
14.4.5	Expression <code>edm:Decimal</code>	52
14.4.6	Expression <code>edm:Duration</code>	53
14.4.7	Expression <code>edm:EnumMember</code>	53
14.4.8	Expression <code>edm:Float</code>	53
14.4.9	Expression <code>edm:Guid</code>	53
14.4.10	Expression <code>edm:Int</code>	54
14.4.11	Expression <code>edm:String</code>	54
14.4.12	Expression <code>edm:TimeOfDay</code>	54
14.5	Dynamic Expressions	54
14.5.1	Comparison and Logical Operators	55
14.5.2	Expression <code>edm:AnnotationPath</code>	55
14.5.3	Expression <code>edm:Apply</code>	56
14.5.4	Expression <code>edm:Cast</code>	57
14.5.5	Expression <code>edm:Collection</code>	58
14.5.6	Expression <code>edm:If</code>	58
14.5.7	Expression <code>edm:IsOf</code>	58
14.5.8	Expression <code>edm:LabeledElement</code>	59
14.5.9	Expression <code>edm:LabeledElementReference</code>	59
14.5.10	Expression <code>edm:Null</code>	59
14.5.11	Expression <code>edm:NavigationPropertyPath</code>	60
14.5.12	Expression <code>edm:Path</code>	60
14.5.13	Expression <code>edm:PropertyPath</code>	62
14.5.14	Expression <code>edm:Record</code>	62
14.5.15	Expression <code>edm:UrlRef</code>	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	72
15.6	Entity Container	73
15.7	Terms and Annotations.....	75
16	CSDL Examples	79
16.1	Products and Categories Example	79
16.2	Annotations for Products and Categories Example.....	81
17	Attribute Values	82
17.1	Namespace.....	82
17.2	SimpleIdentifier	82
17.3	QualifiedName	82
17.4	TypeName	82
17.5	TargetPath	82
17.6	Boolean.....	83
18	Conformance	84
Appendix A.	Acknowledgments	85
Appendix B.	Revision History	86

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] | <i>OData ABNF Construction Rules Version 4.0</i> .
See link in “Additional artifacts” section on cover page. |
| [OData-Atom] | <i>OData ATOM Atom Format Version 4.0</i> .
See link in “Related work” section on cover page. |
| [OData-EDM] | <i>OData EDM XML Schema</i> .
See link in “Additional artifacts” section on cover page. |
| [OData-EDMX] | <i>OData EDMX XML Schema</i> .
See link in “Additional artifacts” section on cover page. |
| [OData-JSON] | <i>OData JSON Format Version 4.0</i> .
See link in “Related work” section on cover page. |
| [OData-Meta] | <i>OData Metadata Service Schema</i> .
See link in “Additional artifacts” section on cover page. |
| [OData-Protocol] | <i>OData Version 4.0 Part 1: Protocol</i> .
See link in “Additional artifacts” section on cover page. |
| [OData-URL] | <i>OData Version 4.0 Part 2: URL Conventions</i> .
See link in “Additional artifacts” section on cover page. |
| [OData-VocCore] | <i>OData Core Vocabulary</i> .
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 `edm: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/edm`

Prior versions of OData used the following namespace for EDMX:

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

They are non-normative for this specification.

In this specification the namespace prefix `edm` 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 `edm:Edmx`

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

Example 2:

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

3.1.1 Attribute `Version`

The `edm: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 `edm:DataServices`

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

3.3 Element `edm:Reference`

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

The `edm: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://vocab.odata.org/capabilities/v1">
    <edmx:Include Namespace="Org.OData.Capabilities.V1" />
  </edmx:Reference>
  <edmx:Reference Uri="http://vocab.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 `edm:IncludeAnnotations`

The `edm:Reference` element contains zero or more `edm:IncludeAnnotations` elements that specify the annotations to include from the target document. If no `edm: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"?>
<edm:Edmx xmlns:edm="http://docs.oasis-open.org/odata/ns/edm"
  Version="4.0">
  <edm:Reference Uri="http://odata.org/ann/b">
    <edm:IncludeAnnotations TermNamespace="org.example.validation" />
    <edm:IncludeAnnotations TermNamespace="org.example.display"
      Qualifier="Tablet" />
    <edm:IncludeAnnotations TermNamespace="org.example.hcm"
      TargetNamespace="com.entoseexample.Sales" />
    <edm:IncludeAnnotations TermNamespace="org.example.hcm"
      Qualifier="Tablet"
      TargetNamespace="com.entoseexample.Person" />
  </edm:Reference>
  <edm:DataServices>...</edm:DataServices>
</edm: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.entoseexample.Sales` namespace and
- Annotations that apply a term from the `org.example.hcm` namespace to an element of the `com.entoseexample.Person` namespace and specify a `Tablet` qualifier.

3.5.1 Attribute `TermNamespace`

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

The `edm: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 `edm: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 `edm: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 `edm: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 `edm: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 no value is specified for a property whose `Type` attribute specifies a collection, the client cannot assume any default value.

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. A `Nullable` value of `true` means that the collection MAY contain null values (although attempting to insert a null value may still fail for a variety of reasons). A `Nullable` value of `false` means that the collection cannot contain null values. The absence of the `Nullable` attribute means it is unknown 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. For binary or stream properties this is the octet length of the binary data, for string property it is the character length of the string value. 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 significant decimal digits allowed inof 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 2928 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. If `Precision` is equal to `Scale`, a single zero has to precede the decimal point.

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.

Note: if the underlying data store allows negative scale, services may use a `Precision` attribute with the absolute value of the negative scale added to the actual number of significant decimal digits, and client-provided values may have to be rounded before being stored.

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.~~
Allowed values: 0.23, 0.7, not allowed values: 1.23, 1.2.

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

Example 11: *Precision* and a variable *Scale* applied to the *Decimal* type.
Allowed values: 0.123, 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*~~The value assigned to this attribute ~~*true*~~ indicates that the value of the property is encoded ~~might~~ contain and accept string values with Unicode. ~~A~~ characters beyond the ASCII character set. The value ~~false~~ value assigned to this attribute indicates that the value of the property is encoded ~~will only contain~~ and accept string values with ASCII characters limited to the ASCII character set.

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 `edm: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 of 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.

Enumeration types can have multiple members with the same value. Members with the same value compare as equal, and members with the same value can be used interchangeably.

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~~ [key predicates](#) appended to the path that identifies the composable function, and with system query options 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="ContextMainSupplier" 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.

Note: [services should not introduce cycles with Extends. Clients should be prepared to process cycles introduced with Extends.](#)

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. Clients SHOULD ignore unknown terms and silently treat unexpected or invalid values (including invalid type, invalid literal expression, etc.) as an unknown value for the term.

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 `IsURL` 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="IsURLIsURL" Type="Core.Tag" DefaultValue="true"
  AppliesTo="Property Term">
  <Annotation Term="Core.Description">
    <String>
      Properties and terms annotated with this term MUST contain a valid
URLURL
    </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:Action` (applies to all overloads)
- `edm:ActionImport`
- `edm:ComplexType`
- `edm:EntityContainer`
- `edm:EntitySet`
- `edm:EntityType`
- `edm:EnumType`
- `edm:Function` (applies to all overloads)
- `edm:FunctionImport`
- `edm:Member`
- `edm:NavigationProperty` (via type, entity set, or singleton)
- `edm:Parameter` (applies to all overloads defining the parameter)
- `edm:Property` (via type, entity set, or singleton)
- `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 do 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 structured type followed by zero or more property, navigation property, or type cast segments, each segment starting with a forward slash`
- `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		
<code>edm:And</code>	Logical and	<code><And><Path>IsMale</Path><Path>IsMarried</Path></And></code>
<code>edm:Or</code>	Logical or	<code><Or><Path>IsMale</Path><Path>IsMarried</Path></Or></code>
<code>edm:Not</code>	Logical negation	<code><Not><Path>IsMale</Path></Not></code>
Comparison Operators		
<code>edm:Eq</code>	Equal	<code><Eq><Null/><Path>IsMale</Path></Eq></code>
<code>edm:Ne</code>	Not equal	<code><Ne><Null/><Path>IsMale</Path></Ne></code>
<code>edm:Gt</code>	Greater than	<code><Gt><Path>Price</Path><Int>20</Int></Gt></code>
<code>edm:Ge</code>	Greater than or equal	<code><Ge><Path>Price</Path><Int>10</Int></Ge></code>
<code>edm:Lt</code>	Less than	<code><Lt><Path>Price</Path><Int>20</Int></Lt></code>
<code>edm:Le</code>	Less than or equal	<code><Le><Path>Price</Path><Int>100</Int></Le></code>

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~~ **following exceptions**:

- The `AnnotationPath` expression may traverse multiple collection-valued structural or navigation properties.
- 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 [Qualified Name](#) 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 `NameOfMovieGenreName` property of the `Actor` entity

```
<Apply Function="odata.fillUriTemplate">
  <String>http://host/service/Genres('{genreName}')</someAPI/Actors/{actorName}/
  CV</String>
  <LabeledElement Name="genreNameactorName" Path="NameOfMovieGenreActor/Name"
  />
</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 only allowed child elements of the null expression ~~MUST NOT contain any other~~ are `edm:Annotation` 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 [or the value of the term cast](#) 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>
```

```
</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" }
  ]
}
```

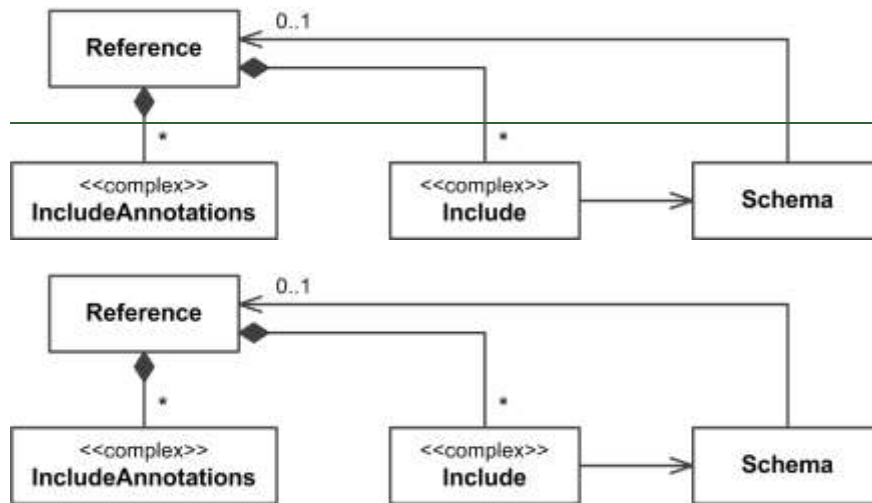
Note: all examples in this chapter use ~/ as an abbreviation for the service root URL.

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

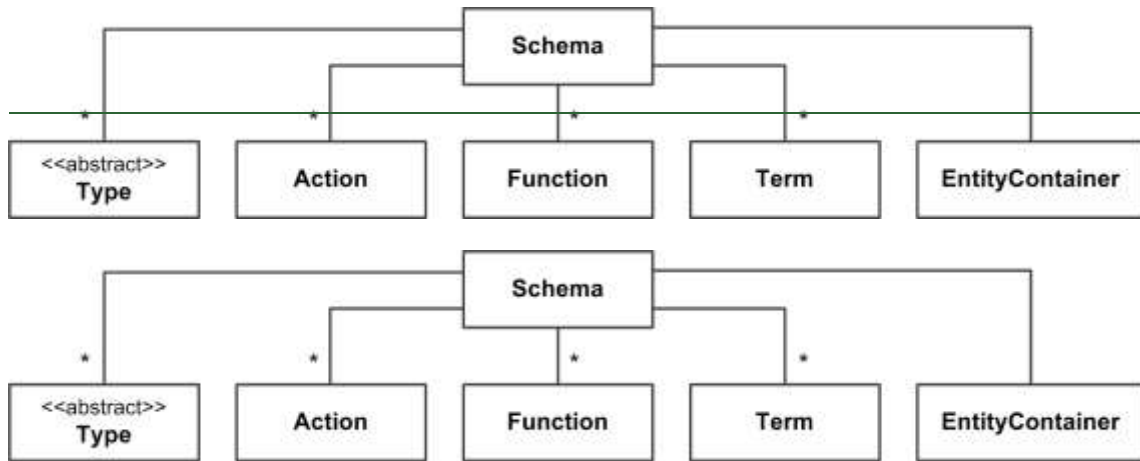
```
GET ~/metadata/References?$expand=Include/Schema($select=Namespace)
```

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')
```

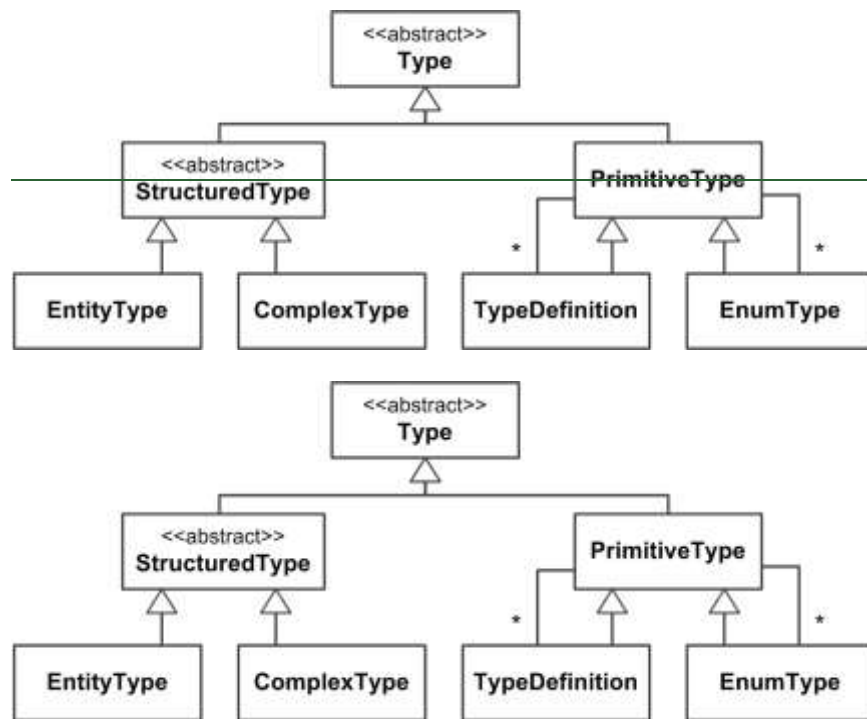
would return

```
{
  "@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
  }, {
```

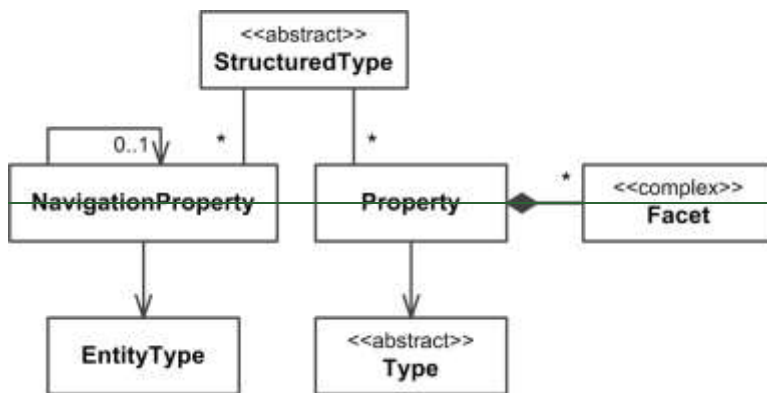
```

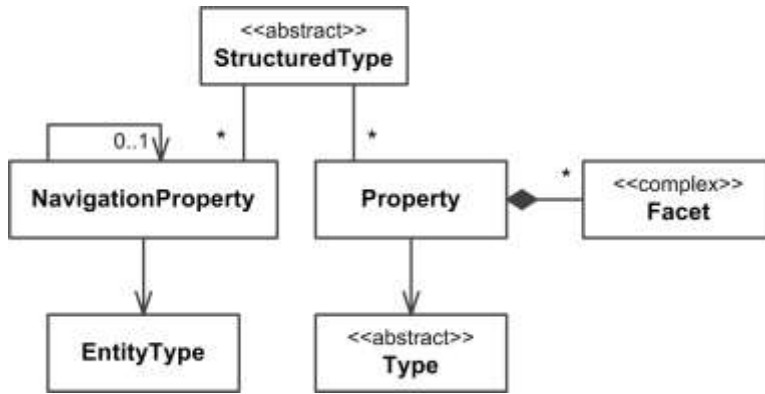
"@odata.type":"Meta.EntityType",
"QualifiedName":"ODataDemo.Supplier", "Name":"Supplier",
"Key":[{"PropertyPath":"ID","Alias":null}],
"Abstract":false, "OpenType":false, "HasStream":false
},{
"@odata.type":"Meta.EntityType",
"QualifiedName":"ODataDemo.Country", "Name":"Country",
"Key":[{"PropertyPath":"Code","Alias":null}],
"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":[]
  }
}

```

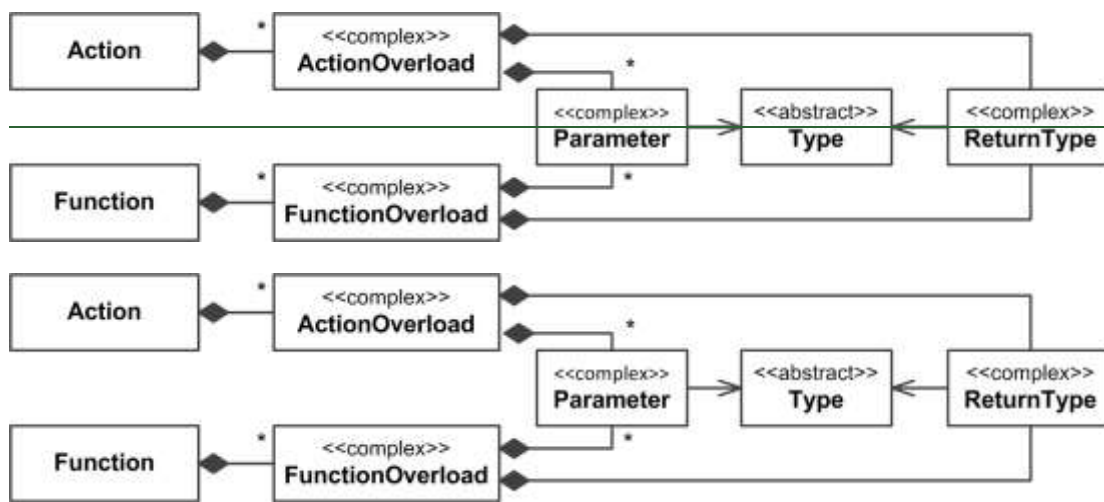
```

    {
      "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" }
          }
        }
      ]
    }
  ]
}

```



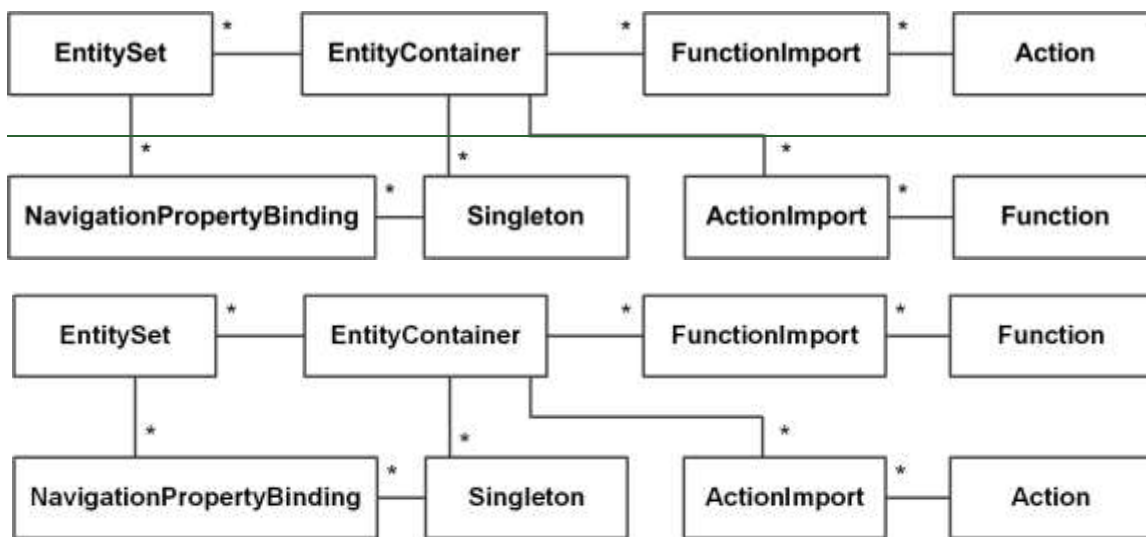
```

    },
    "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

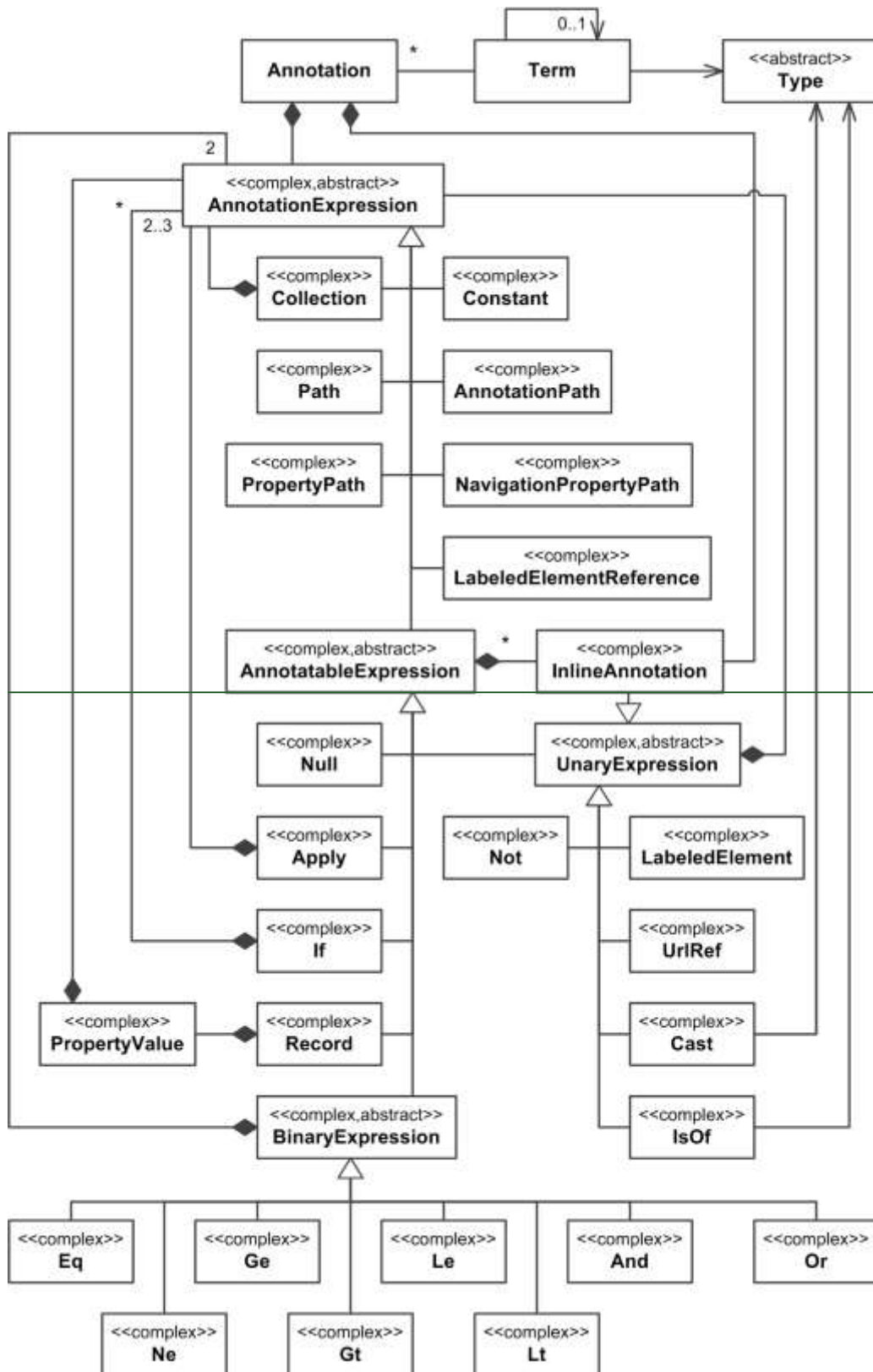
```
GET ~/metadata/EntityContainer?$expand=*
```

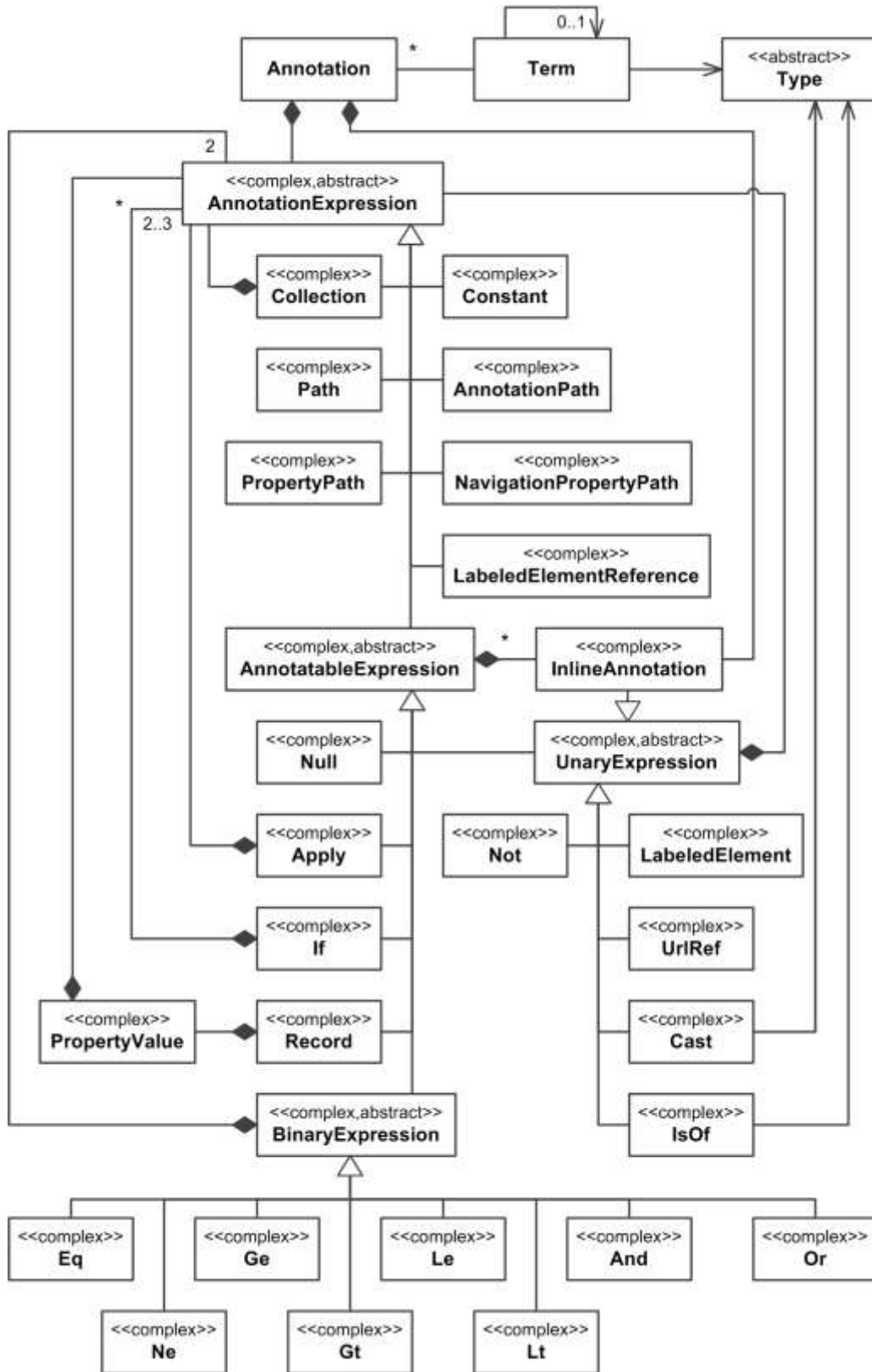
would return

```
{
  "@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", "Fullname": "ODataDemo.DemoService/Contoso", "Name": "Contoso",
"Name": "Contoso", "MainSupplier"
        }
      ],
      "ActionImports": [],
      "FunctionImports": [
        {
"QualifiedName": "ODataDemo.DemoService/ProductsByRating", "Fullname": "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>
  </Schema>
  </DataServices>
  </Edmx>
```

```

    </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="ContosoMainSupplier" 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 annotations 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, Martin Zurmuehl	Repaired mechanical error in the editable source
<u>Errata 03</u>	<u>2016-03-03</u>	<u>Michael Pizzo,</u> <u>Ralf Handl,</u> <u>Martin Zurmuehl</u>	<u>Minor changes and clarifications</u>