OData Common Schema Definition Language (CSDL) XML Representation Version 4.01

Committee Specification Draft ~~01~~02 / Public Review Draft ~~01~~02

08 ~~December 2016~~June 2017

Specification URIs

This version:

http://docs.oasis-open.org/odata/odata-csdl-xml/v4.01/csprd02/odata-csdl-xml-v4.01-csprd02.docx (Authoritative)
http://docs.oasis-open.org/odata/odata-csdl-xml/v4.01/csprd02/odata-csdl-xml-v4.01-csprd02.html
http://docs.oasis-open.org/odata/odata-csdl-xml/v4.01/csprd02/odata-csdl-xml-v4.01-csprd02.pdf

Previous version:

http://docs.oasis-open.org/odata/odata-csdl-xml/v4.01/csprd01/odata-csdl-xml-v4.01-csprd01.docx (Authoritative)
http://docs.oasis-open.org/odata/odata-csdl-xml/v4.01/csprd01/odata-csdl-xml-v4.01-csprd01.html
http://docs.oasis-open.org/odata/odata-csdl-xml/v4.01/csprd01/odata-csdl-xml-v4.01-csprd01.pdf

~~**Previous version:**~~

~~N/A~~

Latest version:

http://docs.oasis-open.org/odata/odata-csdl-xml/v4.01/odata-csdl-xml-v4.01.docx (Authoritative)
http://docs.oasis-open.org/odata/odata-csdl-xml/v4.01/odata-csdl-xml-v4.01.html
http://docs.oasis-open.org/odata/odata-csdl-xml/v4.01/odata-csdl-xml-v4.01.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~~
Ralf Handl (ralf.handl@sap.com), SAP SE
Michael Pizzo (mikep@microsoft.com), Microsoft

Editors:

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

Additional artifacts:

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

- XML schemas: OData EDMX XML Schema and OData EDM XML Schema. ~~http://docs.oasis-open.org/odata/odata-csdl-xml/v4.01/csprd01/schemas/.~~http://docs.oasis-open.org/odata/odata-csdl-xml/v4.01/csprd02/schemas/.

Related work:

This specification replaces or supersedes ~~the parts related to the XML representation previously specified in~~:

- *OData Version 4.0 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. Latest version: ~~-~~ http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part3-csdl.html~~.~~

This specification is related to:

- *OData Version 4.01*. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. A multi-part Work Product which includes:
  - *OData Version 4.01. Part 1: Protocol*. Latest version: http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.01-part1-protocol.html.
  - *OData Version 4.01. Part 2: URL Conventions*. Latest version: http://docs.oasis-open.org/odata/odata/v4.01/odata-v4.01-part2-url-conventions.html.
  - *ABNF components: OData ABNF Construction Rules Version 4.01 and OData ABNF Test Cases*. http://docs.oasis-open.org/odata/odata/v4.01/csprd02/abnf/.
- *OData Common Schema Definition Language (CSDL) JSON Representation Version 4.01.* Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. Latest version: http://docs.oasis-open.org/odata/odata-csdl-json/v4.01/odata-csdl-json-v4.01.html.
- *OData Vocabularies Version 4.0.* Edited by ~~Mike~~Michael Pizzo, Ralf Handl, and Ram Jeyaraman. Latest version: http://docs.oasis-open.org/odata/odata-vocabularies/v4.0/odata-vocabularies-v4.0.html
- ~~*OData Version 4.01*. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. A multi-part Work Product which includes:~~
  - ~~*OData Version 4.01. Part 1: Protocol*. Latest version: http://docs.oasis-open.org/odata/odata/v4.01/odata-v4.01-part1-protocol.html.~~
  - ~~*OData Version 4.01. Part 2: URL Conventions*. Latest version: http://docs.oasis-open.org/odata/odata/v4.01/odata-v4.01-part2-url-conventions.html.~~
  - ~~*OData Version 4.01. Part 3: Common Schema Definition Language (CSDL)*. Latest version: http://docs.oasis-open.org/odata/odata/v4.01/odata-v4.01-part3-csdl.html.~~
- *OData JSON Format Version 4.01*. Edited by ~~Ralf Handl,~~ Michael Pizzo, Ralf Handl, and Mark Biamonte. Latest version: ~~-~~ http://docs.oasis-open.org/odata/odata-json-format/v4.01/odata-json-format-v4.01.html.
- ~~*OData Extension for Data Aggregation Version 4.0*. Edited by Ralf Handl, Hubert Heijkers, Gerald Krause, Michael Pizzo, and Martin Zurmuehl. Latest version: http://docs.oasis-open.org/odata/odata-data-aggregation-ext/v4.0/odata-data-aggregation-ext-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 specific representations of the entity data model exposed by an OData service using, XML, JSON, and other formats. This document (OData CSDL XML Representation) specifically defines the XML representation of ~~the entity data model. This XML representation is based on XML Schema~~CSDL.

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/.

This Committee Specification Public Review Draft is provided under the RF on RAND Terms Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the ~~Technical Committee web page (https://www.oasis-open.org/committees/odata/ipr.php).~~TC's web page (https://www.oasis-open.org/committees/odata/ipr.php).

Note that any machine-readable content (aka Computer Language Definitions) declared Normative for this Work Product is provided in separate plain text files. In the event of a discrepancy between any such plain text file and display content in the Work Product's prose narrative document(s), the content in the separate plain text file prevails.

Citation format:

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

[OData-CSDL-XML-v4.01]

OData Common Schema Definition Language (CSDL) XML Representation Version 4.01. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. 08 ~~December 2016.~~June 2017. OASIS Committee Specification Draft ~~01~~02 / Public Review Draft ~~01~~02. ~~http://docs.oasis-open.org/odata/odata-csdl-xml/v4.01/csprd01/odata-csdl-xml-v4.01-csprd01.html~~http://docs.oasis-open.org/odata/odata-csdl-xml/v4.01/csprd02/odata-csdl-xml-v4.01-csprd02.html. Latest version: http://docs.oasis-open.org/odata/odata-csdl-xml/v4.01/odata-csdl-xml-v4.01.html.

Notices

Table of Contents

1 Introduction

1 Introduction

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

1.0 IPR Policy

This Working Draft is being developed under the RF on RAND Terms Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC's web page (https://www.oasis-open.org/committees/odata/ipr.php).

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~~

1.2 Normative References

[ECMAScript] *ECMAScript 2016 Language Specification, 7th Edition,* June 2016. Standard ECMA-262. http://www.ecma-international.org/publications/standards/Ecma-262.htm.

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

[OData-ABNF] *OData ABNF Construction Rules Version 4.01*.
See link in "Additional artifacts" section on cover page.

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

[OData-EDMX] *OData EDMX XML Schema*.
See link in "Additional artifacts" section on cover page.

[OData-CSDLJSON] *OData Common Schema Definition Language (CSDL) JSON Representation Version 4.01*. See link in "Related work" section on cover page.

[OData-JSON] *OData JSON Format Version 4.01.*
See link in "Related work" section on cover page.

[OData-Protocol] *OData Version 4.01 Part 1: Protocol*.
See link in "Additional artifacts" section on cover page.

[OData-URL] *OData Version 4.01 Part 2: URL Conventions*.
See link in "Additional artifacts" section on cover page.

[OData-VocCore] *OData Vocabularies Version 4.0: Core Vocabulary.*
See link in "Related work" section on cover page.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997. https://tools.ietf.org/html/rfc2119.

[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~~

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.

Representation-specific text is indented and marked with vertical lines.

> **Representation-Specific Headline**
>
> Normative representation-specific text

All other text is normative unless otherwise labeled.

2 CSDL Namespaces

2 XML Representation

OData CSDL XML is a full representation of the OData Common Schema Definition Language 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]**.

It is an alternative to the CSDL JSON representation defined in **[OData-CSDLJSON]** and neither adds nor removes features.

2.1 Requesting the XML Representation

The OData CSDL XML representation can be requested using the `$format` query option in the request URL with the media type `application/xml`, optionally followed by media type parameters, or the case-insensitive abbreviation `xml` which MUST NOT be followed by media type parameters.

Alternatively, this representation can be requested using the `Accept` header with the media type `application/xml`, optionally followed by media type parameters.

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

The response MUST contain the `Content-Type` header with a value of `application/xml`, optionally followed by media type parameters.

This specification does not define additional parameters for the media type `application/xml`.

2.2 XML Namespaces

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

~~2.1 Namespace EDMX~~

2.2.1 Namespace EDMX

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

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

Prior versions of OData used the following namespace for EDMX:

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

They are non-normative for this specification.

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

~~2.2 Namespace EDM~~

2.2.2 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~~

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

~~2.4 XML Document Order~~

2.4 XML Document Order

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

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

~~3 Entity Model Wrapper~~

3 Entity Model

An OData service exposes a single entity model. This model may be distributed over several ~~schemas,~~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~~entity container that defines the resources exposed by this service. This entity container MAY ~~extend~~extend an entity container defined in a referenced ~~documents~~document.

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

3.1 Element ~~edmx:Edmx~~

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

~~*Example 2:*~~

```
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx"
           Version="4.01">
  <edmx:DataServices>
    ...
  </edmx:DataServices>
</edmx:Edmx>
```

3.1.1 Attribute ~~Version~~

~~The `edmx:Edmx` element MUST contain the `Version` attribute to specify the version of the EDMX wrapper returned by the service. For OData 4.0 responses the value of this attribute MUST be `4.0`. For OData 4.01 responses the value of this attribute MUST be `4.01`. Services MUST return a 4.0 response if the request was made with an `OData-MaxVersion` header with a value of `4.0`.~~

3.2 Element ~~edmx:DataServices~~

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

3.3 Element ~~edmx:Reference~~

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

~~The `edmx:Reference` element MAY include the `Core.SchemaVersion` annotation, defined in **[OData-VocCore]**, to indicate a particular version of the referenced schema. If the `Core.SchemaVersion` annotation is present, the `SchemaVersion` header, defined **[OData-Protocol]**, SHOULD be used when retrieving the referenced schema document.~~

~~The `edmx:Reference` element MAY contain zero or more `edm:Annotation` elements.~~

The *scope* of a CSDL document is the document itself and all schemas ~~included~~included from directly ~~referenced documents.~~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.~~ qualified names. This includes the built-in primitive and abstract types.

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~~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: references to entity models containing definitions of vocabulary terms~~

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

~~3.3.1 Attribute `Uri`~~

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

~~3.4 Element `edmx:Include`~~

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

~~3.4.1 Attribute `Namespace`~~

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

~~3.4.2 Attribute `Alias`~~

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

resolved to a fully qualified name by examining aliases on `edmx:Include` and `edm:Schema` elements within the same document.

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

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

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

3.5 Element `edmx:IncludeAnnotations`

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

Example 4: reference documents that contain annotations

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

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

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

3.1 Nominal Types

3.5.1 Attribute `TermNamespace`

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

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

3.5.2 Attribute `Qualifier`

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

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

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

3.5.3 Attribute `TargetNamespace`

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

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

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.simple identifier. Nominal types are referenced using their QualifiedNamequalified name. 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

3.2 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 typesEntity 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

Structured Types are composed of zero or more structural properties and navigation properties.

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

3.3 Primitive Types

- 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
Edm.DateTimeOffset	Date and time with a time-zone offset, no leap seconds
Edm.Decimal	Numeric values with decimal representation
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

Type	Meaning
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.

All numeric types allow the special numeric values -INF, INF, and NaN, and Edm.Date and Edm.DateTimeOffset allow the special values -INF, INF.

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

~~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~~facets, defined in section "Type Facets".

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

3.4 Built-In Abstract Types

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

- Edm.PrimitiveType
- Edm.ComplexType
- Edm.EntityType
- Edm.Untyped

Conceptually, these are the abstract base types for primitive types (including type definitions and enumeration types), complex types, entity types, or any type or collection of 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.
- Edm.Untyped
  - cannot be returned in a payload with an OData-Version header of 4.0. Services should treat untyped properties as dynamic properties in 4.0 payloads.
  - cannot be used as the type of a key property of an entity type.
  - cannot be the base type of an entity type or complex type.
  - cannot be used as the underlying type of a type definition or enumeration type.
- Collection(Edm.PrimitiveType)

- cannot be used as the type of a property.
- cannot be used as the return type of a function.

- `Collection(Edm.Untyped)`
  - cannot be returned in a payload with an `OData-Version` header of `4.0`. Services should treat untyped properties as dynamic properties in `4.0` payloads.

3.5 Built-In Types for defining Vocabulary Terms

Vocabulary terms can, in addition, use

- `Edm.AnnotationPath`
- `Edm.PropertyPath`
- `Edm.NavigationPropertyPath`
- `Edm.AnyPropertyPath` (`Edm.PropertyPath` or `Edm.NavigationPropertyPath`)

~~7.2.~~

~~See rule `primitiveLiteral` in **[OData-ABNF]** for the representation of primitive type values in URLs 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`~~
- ~~`Edm.Untyped`~~

~~Conceptually, these are the abstract base types for primitive types (including type definitions and enumeration types), complex types, entity types, or any type or collection of 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.~~
- ~~`Edm.Untyped`~~
  - ~~cannot be returned in a payload with an `OData-Version` header of `4.0`. Services should treat untyped properties as dynamic properties in `4.0` payloads.~~
  - ~~cannot be used as the type of a key property of an entity type.~~
  - ~~cannot be the base type of an entity type or complex type.~~
  - ~~cannot be used as the underlying type of a type definition or enumeration type.~~
- ~~`Collection(Edm.PrimitiveType)`~~

- cannot be used as the type of a property.
- cannot be used as the return type of a function.

- `Collection(Edm.Untyped)`
  - cannot be returned in a payload with an `OData-Version` header of `4.0`. Services should treat untyped properties as dynamic properties in `4.0` payloads.

Vocabulary terms can, in addition, use

- `Edm.AnnotationPath`
- `Edm.PropertyPath`
- `Edm.NavigationPropertyPath`
- `Edm.AnyPropertyPath` (`Edm.PropertyPath` or `Edm.NavigationPropertyPath`)
- `Edm.AnyPath` (`Edm.AnyPropertyPath` or `Edm.ModelElementPath` (any model element, including `Edm.AnnotationPath`, `Edm.NavigationPropertyPath`, and `Edm.PropertyPath`)

as the type of a primitive term, or the type of a property of a complex type (recursively) that is exclusively used as the type of a term. See section "Path Expressions" for details.

4.6 Annotations

3.6 Annotations

Many parts of the model can be annotateddecorated with additional information using the `edm:Annotation`annotations. Annotations are identified by their term name and an optional qualifier that allows applying the same term multiple times to the same model element.

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

4 ~~Vocabulary annotations can be specified as a child of the model~~CSDL XML Document

Element edmx:Edmx

The edmx:Edmx element ~~being annotated or as a child of an edm:Annotations~~is the root element ~~that targets~~of a CSDL XML document. It MUST contain the ~~model~~Version attribute and it MUST contain exactly one edmx:DataServices element.

It MAY contain edmx:Reference elements to reference other CSDL documents.

Attribute Version

The Version attribute specifies the OData protocol version of the service. For OData 4.0 responses the value of this attribute MUST be 4.0. For OData 4.01 responses the value of this attribute MUST be 4.01. ~~Refer to Vocabulary Annotations for details on~~Services MUST return an OData 4.0 response if the request was made with an OData-MaxVersion header with a value of 4.0.

Element edmx:DataServices

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

Example 2:

```
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx"
           Version="4.01">
  <edmx:DataServices>
    …
  </edmx:DataServices>
</edmx:Edmx>
```

4.1 Reference

A reference to an external CSDL document allows to bring part of the referenced document's content into the scope of the referencing document.

A reference MUST specify a URI that uniquely identifies the referenced document, so two references MUST NOT specify the same URI. The URI SHOULD be a URL that locates the referenced document. If the URI is not dereferencable it SHOULD identify a well-known schema. The URI MAY be absolute or relative URI; relative URLs are relative to the URL of the document containing the reference, or relative to a base URL specified in a format-specific way.

A reference MAY be annotated.

The Core.SchemaVersion annotation, defined in **[OData-VocCore]**, MAY be used to indicate a particular version of the referenced schema. If the Core.SchemaVersion annotation is present, the SchemaVersion header, defined **[OData-Protocol]**, SHOULD be used when retrieving the referenced schema document.

Element edmx:Reference

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

The `edmx:Reference` element MUST contain the `Uri` attribute, and it MUST contain at least one `edmx:Include` or `edmx:IncludeAnnotations` child element.

It MAY contain `edm:Annotation` elements.

Attribute `Uri`

The value of `Uri` is an absolute or relative URI; relative URIs are relative to the `xml:base` attribute, see **[XML-Base]**.

Example 3: references to other CSDL documents

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx"
           Version="4.0">
  <edmx:Reference Uri="http://vocabs.odata.org/capabilities/v1">
    …
  </edmx:Reference>
  <edmx:Reference Uri="http://vocabs.odata.org/core/v1">
    …
  </edmx:Reference>
  <edmx:Reference Uri="http://example.org/display/v1">
    …
  </edmx:Reference>
  <edmx:DataServices>…</edmx:DataServices>
</edmx:Edmx>
```

4.2 Included Schema

A reference MAY include zero or more schemas from the referenced document.

The included schemas are identified via their namespace. The same namespace MUST NOT be included more than once, even if it is declared in more than one referenced document.

When including a schema, a simple identifier value MAY be specified as an alias for the schema that is used in qualified names instead of the namespace. For example, 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 for included schemas and schemas defined within the document.

If an included schema specifies an alias, the alias MAY be used instead of the namespace within qualified names to identify model elements of the included schema. An alias only provides a more convenient notation, allowing a short string to be substituted for a long namespace. Every model element that can be identified via an alias-qualified name can alternatively be identified via its full namespace-qualified name.

Aliases are document-global, so all schemas defined within or included into a document MUST have different aliases.

The alias MUST NOT be one of the reserved values `Edm`, `odata`, `System`, or `Transient`.

An alias is only valid within the document in which it is declared; a referencing document may define its own aliases for included schemas.

Element `edmx:Include`

The `edmx:Include` element specifies a schema to include from the referenced CSDL document. It MUST provide the `Namespace` attribute and it MAY provide the `Alias` attribute.

It MAY contain `edm:Annotation` elements.

Attribute `Namespace`

The value of `Namespace` is the namespace of a schema defined in the referenced CSDL document.

Attribute `Alias`

The value of `Alias` is a simple identifier that can be used in qualified names instead of the namespace.

Example 4: references to entity models containing definitions of vocabulary terms

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

4.3 Included Annotations

In addition to including whole schemas with all model constructs defined within that schema, annotations can be included with more flexibility.

Annotations are selectively included by specifying the namespace of the annotations' term. Consumers can opt not to inspect the referenced document if none of the term namespaces is of interest for the consumer.

In addition, the qualifier of annotations to be included MAY be specified. For instance, a service author might want to supply a different set of annotations for various device form factors. If a qualifier is specified, only those annotations from the specified term namespace with the specified qualifier (applied to a model element of the target namespace, if present) SHOULD be included. If no qualifier is specified, all annotations within the referenced document from the specified term namespace (taking into account the target namespace, if present) SHOULD be included.

The qualifier also provides consumers insight about what qualifiers are present in the referenced document. If the consumer is not interested in that particular qualifier, the consumer can opt not to inspect the referenced document.

In addition, the namespace of the annotations' target MAY be specified. If a target namespace is specified, only those annotations which apply a term form the specified term namespace to a model element of the target namespace (with the specified qualifier, if present) SHOULD be included. If no target namespace is specified, all annotations within the referenced document from the specified term namespace (taking into account the qualifier, if present) SHOULD be included.

The target namespace also provides consumers insight about what namespaces are present in the referenced document. If the consumer is not interested in that particular target namespace, the consumer can opt not to inspect the referenced document.

Element `edmx:IncludeAnnotations`

The `edmx:IncludeAnnotations` element specifies the annotations to include from the referenced CSDL document. If no `edmx:IncludeAnnotations` element is specified, a client MAY ignore all annotations in the referenced document that are not explicitly used in an `edm:Path` expression of the referencing document.

The `edmx:IncludeAnnotations` element MUST provide the `TermNamespace` attribute, and it MAY provide the `Qualifier` and `TargetNamespace` attribute.

Attribute TermNamespace

The value of TermNamespace is a namespace.

Attribute Qualifier

The value of Qualifier is a simple identifier.

Attribute TargetNamespace

The value of TargetNamespace is a namespace.

Example 5: reference documents that contain annotations

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

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

- *Annotations that use a term from the `org.example.validation` namespace, and*
- *Annotations that use a term from the `org.example.display` namespace and specify a `Tablet` qualifier and*

~~5~~ Annotations that apply a term from the `org.example.hcm` namespace to an element of the `com.example.Sales` ~~Schema~~

- *namespace and*
- *Annotations that apply a term from the `org.example.hcm` namespace to an element of the `com.example.Person` namespace and specify a `Tablet` qualifier.*

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 namespacenamespace. All edm:Schema elements MUST have a namespace defined through a Namespace attribute whichnamespaces MUST be unique within the scope of a 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 namesqualified 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 namespace MUST NOT usebe one of the reserved values Edm, odata, System, or Transient.

> **Element edm:Schema**
>
> The edm:Schema element defines a schema. It MUST contain the Namespace attribute and it MAY contain the Alias attribute.
>
> It MUST contain one or more of the elements edm:Action, edm:Annotations, edm:Annotation, edm:ComplexType, edm:EntityContainer, edm:EntityType, edm:EnumType, edm:Function, edm:Term, or edm:TypeDefinition.
>
> #### 5.1.2 Attribute AliasNamespace
>
> The value of Namespace is the namespace of the schema

5.1 Alias

A schema MAY ~~define~~specify an alias ~~by providing a SimpleIdentifier value for the `Alias` attribute. An alias allows nominal types to be~~ which MUST be a simple identifier.

> If a schema specifies an alias, the alias MAY be used instead of the namespace within qualified ~~with~~names to identify model elements of that schema. An alias only provides a more convenient notation, allowing a short string ~~rather than~~to be substituted for a long namespace. Every model element that can be identified via an alias-qualified name can alternatively be identified via its full namespace-qualified name.

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

The ~~`Alias` attribute~~ alias MUST NOT ~~use~~be one of the reserved values `Edm`, `odata`, `System`, or `Transient`.

> ### Attribute `Alias`
>
> The value of `Alias` is a simple identifier.

Example 6: schema `org.example` with an alias and a description for the schema

```
<Schema Namespace="org.example" Alias="self">
  <Annotation Term="Core.Description" String="Example schema" />
  …
</Schema>
```

5.2 Annotations with External Targeting

> ### Element `edm:Annotations`

~~6~~ The `edm:Annotations` element is used to apply a group of annotations to a single model element. ~~Entity Type~~

It MUST contain the `Target` attribute and it MAY contain the `Qualifier` attribute.

It MUST contain at least one `edm:Annotation` element.

Attribute `Target`

The value of `Target` is a path expression identifying the annotation target. It MUST resolve to a model element in scope.

Attribute `Qualifier`

The value of `Qualifier` is a simple identifier.

Example 7: annotations should only be applied to tablet devices

```
<Annotations Target="org.example.Person" Qualifier="Tablet">
  …
</Annotations>
```


6 Entity Type

Entity types are ~~nominal~~nominal ~~structured types~~structured types with a key that consists of one or more references to ~~structural properties~~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.~~

The entity type's name is a simple identifier that MUST be unique within its schema.

An entity type can define two types of properties. ~~A structural property~~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~~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.~~

Element `edm:EntityType`

The `edm:EntityType` element MUST contain the `Name` attribute, and it MAY contain the `BaseType`, `Abstract`, `OpenType`, and `HasStream` attributes.

It MAY contain `edm:Property` and `edm:NavigationProperty` elements describing the properties of the entity type.

It MAY contain one `edm:Key` element.

It MAY contain `edm:Annotation` elements.

Attribute `Name`

The value of `Name` is the entity type's name.

Example 8: 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~~self.Manager" />
</EntityType>
```

6.1 Derived Entity Type

An entity type can inherit from another entity type by specifying it as its base type.

An entity type inherits the key as well as structural and navigation properties of its base type.

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

Attribute `BaseType`

The value of `BaseType` is the qualified name of the base type.

Example 9: a derived entity type based on the previous example


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

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

6.1 Element edm:EntityType

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

It MAY contain one edm:Key element.

6.1.1 Attribute Name

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

6.1.2 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.

6.1.3 Attribute Abstract

6.2 Abstract Entity Type

An entity type MAY indicate that it is abstract and cannot be instantiated by providing a Boolean value of true to the Abstract attribute. If not specified, the Abstract attribute defaults to falsehave instances.

For OData 4.0 responses with an edmx version attribute of 4.0, if Abstract is false, thea non-abstract entity type MUST define a keykey or derive from a base typebase type with a defined key.

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

6.1.4 Attribute OpenTypeAbstract

The value of Abstract is one of the Boolean literals true or false. Absence of the attribute means false.

6.3 Open Entity Type

An entity type MAY indicate that it is open by providing a value of true for the OpenType attribute. An open typeand allows clients to add properties dynamically to instances of the type by specifying uniquely named property 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 attributeindicate that it is also open.

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]**.

~~6.1.5~~ Attribute ~~HasStream~~OpenType

The value of `OpenType` is one of the Boolean literals `true` or `false`. Absence of the attribute means `false`.

6.4 Media Entity Type

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

~~A value of `true` specifies~~ that ~~the entity type~~ it is a media entity type. *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~~An entity type derived ~~types.~~

~~Entity types~~from a media entity type MUST indicate that ~~specify `HasStream="true"`~~it is also a media entity type.

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

~~6.2 Element `edm:Key`~~

Attribute `HasStream`

The value of `HasStream` is one of the Boolean literals `true` or `false`. Absence of the attribute means `false`.

6.5 Key

An entity is uniquely identified within an entity set by its key. A key MAY be specified if the entity type does not specify a base type that already has a key declared.

In order to be specified as the type of an ~~entity set~~entity set or a collection-valued ~~containment navigation property~~containment navigation property, the entity type MUST either ~~contain exactly one `edm:Key` element~~specify a key or inherit its key from its ~~base type.~~base type.

In OData 4.01 responses entity types used for ~~singletons~~singletons or single-valued ~~navigation properties~~navigation properties do not require ~~keys. For~~a key. In OData 4.0 responses ~~with an edmx `version` attribute of `4.0`,~~ entity types used for ~~singletons, entity sets, or navigation properties~~singletons or single-valued navigation properties MUST have a key defined.

An entity type (whether or not it is marked as abstract) MAY define a key only 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 key MUST consist of at least one property.

~~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 NOT be ~~non-~~nullable and MUST be typed with an ~~enumeration type~~enumeration type, one of the following ~~primitive types~~primitive types, or a ~~type definition~~type definition based on one of these ~~primitive types:~~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~~Key property values 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.

A key property MUST be a non-nullable primitive property of the entity type itself or to a non-nullable primitive property of a single-valued, non-nullable complex or navigation property (recursively) of the entity type. Navigation properties MAY only be used in OData 4.01 responses.

If the key property is a property of a complex or navigation property (recursively), the key MUST specify an alias for that property that MUST be a simple identifier and MUST be unique within the set of aliases, structural and navigation properties of the containing entity type and any of its base types.

An alias MUST NOT be defined if the key property is a primitive property of the entity type itself.

For key properties that are a property of a complex or navigation property, the alias MUST be used in the key predicate of URLs instead of the path to the property because the required percent-encoding of the forward slash separating segments of the path to the property would make URL construction and parsing rather complicated. The alias MUST NOT be used in the query part of URLs, where paths to properties don't require special encoding and are a standard constituent of expressions anyway.

Element `edm:Key`

The `edm:Key` element MUST contain at least one `edm:PropertyRef` element.

Element `edm:PropertyRef`

The `edm:PropertyRef` element MUST contain the `Name` attribute and MAY contain the `Alias` attribute.

Attribute `Name`

The value of `Name` is a path expression leading to a primitive property. The names of the properties in the path are joined together by forward slashes.

Attribute `Alias`

The value of `Alias` is a simple identifier.

Example 10: 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 11: entity type with a simple key referencing a property of a ~~complex type~~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 12: 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>
```

6.3 Element edm:PropertyRef

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

6.3.1 Attribute Name

The edm:PropertyRef element MUST specify a value for the Name attribute which MUST be a path expression resolving to a non-nullable primitive property of the entity type itself or to a non-nullable primitive property of a single-valued, non-nullable complex or navigation property (recursively) of the entity type. The names of the properties in the path are joined together by forward slashes. Navigation properties MAY only be used in OData 4.01 responses.

6.3.2 Attribute Alias

If the property identified by the Name attribute is a property of a complex or navigation property, 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 a primitive property of the entity type itself.

For key properties that are a property of a complex or navigation property, the alias MUST be used in the key predicate of URLs instead of the value assigned to the Name attribute because the required percent-encoding of the forward slash separating segments of the path to the property would make URL construction and parsing rather complicated. The alias MUST NOT be used in the query part of URLs, where paths to properties don't require special encoding and are a standard constituent of expressions anyway.

Example 13 (based on example 11): requests to an entity set `Categories` of type `Category` must use the alias

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

Example 14 (based on example 11): in a query part the value assigned to the name attribute must be used

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


7 Structural Property

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

~~7 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).~~

A structural property MUST specify a unique name as well as a type.

The property's name MUST be a simple identifier used when referencing, serializing or deserializing the property. It MUST be unique within the set of structural and navigation properties of the declaring structured type, and MUST NOT match the name of any navigation property in any of its base types. If a structural property with the same name is defined in any of this type's base types, then the property's type MUST be a type derived from the type specified for the property of the base type, and constrains this property to be of the specified subtype for instances of this structured type. The name MUST NOT match the name of any structural or navigation property of any of this type's base types for OData 4.0 responses.

Element `edm:Property`

The `edm:Property` element MUST contain the `Name` and the `Type` attribute, and it MAY contain the facet attributes `Nullable`, `MaxLength`, `Unicode`, `Precision`, `Scale`, `SRID`, and `DefaultValue`.

It MAY contain `edm:Annotation` elements.

Attribute `Name`

The value of `Name` is the property's name.

Example 15: 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.~~

~~7.1 Element `edm:Property`~~

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

~~*Example 14: 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.~~

7.1 Type

The property's type MUST be a primitive type, complex type, or enumeration type in scope, or a collection of one of these types.

A collection-valued property may be annotated with the `Core.Ordered` term, defined in **[OData-CoreVoc]**), to specify that it supports a stable ordering.

A collection-valued property may be annotated with the `Core.PositionalInsert` term, defined in **[OData-CoreVoc]**), to specify that it supports inserting ~~iems~~items into a specific ordinal position.

~~7.1.1~~ Attribute ~~Name~~Type

~~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 defined in the containing structured type, and MUST NOT match the name of any navigation property in any of its base types. If a structural property with the same name is defined in any of this type's base types, then the value of the Type attribute of this property MUST specify a type derived from the type specified for the property of the base type, and constrains this property to be of the specified subtype for instances of this structured type. The name MUST NOT match the name of any structural or navigation property of any of this type's base types for responses with an edmx version attribute of 4.0.~~

~~7.1.2 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.~~

~~7.2 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~~ For single-valued properties the value of Type is the qualified name of the property's type.

For collection-valued properties the value of Type is the character sequence Collection(followed by the qualified name of the property's item type, followed by a closing parenthesis).

Example 16*: property Units that can have zero or more strings as its value*

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

7.2 Type Facets

Facets modify or constrain the acceptable values of a property.

For single-valued properties facets apply to the type ~~referenced in the element where the facet attribute is declared. If the type is a~~of the property. For collection~~,~~-valued properties the facets apply to the type of the items in the collection.

~~*Example 15: Precision facet applied to the DateTimeOffset type*~~

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

~~7.2.1 Attribute Nullable~~

7.2.1 ~~The edm:Property element MAY contain the Nullable attribute whose Boolean value specifies~~Nullable

A Boolean value specifying whether a value is required for the property.

Attribute Nullable

The value of Nullable is one of the Boolean literals true or false.

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

In OData 4.01 responses a ~~property whose `Type` attribute specifies a~~ collection~~-~~-valued property MUST specify a value for the `Nullable` attribute.

If no value is specified for a ~~property whose `Type` attribute specifies a~~ collection-valued property, the client cannot assume any default value. Clients SHOULD be prepared for this situation even in OData 4.01 responses.

~~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 items 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.~~

~~7.2.2 Attribute `MaxLength`~~

7.2.2 ~~A binary, stream or string property MAY define a~~MaxLength

A positive integer value ~~for the `MaxLength` facet attribute. The value of this attribute specifies~~specifying the maximum length of ~~the~~a binary, stream or string value~~ of the property on a type instance~~. For binary or stream ~~properties~~values this is the octet length of the binary data, for string ~~property~~values it is the character~~ length of the string value. Instead of an integer value the constant `max` MAY be~~ .

If no maximum length is specified, clients SHOULD expect arbitrary length.

Attribute `MaxLength`

The value of `MaxLength` is a positive integer or the symbolic value `max` as a shorthand for the maximum length supported for the type by the service.

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

~~7.2.3 Attribute `Precision`~~

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

Note: the symbolic value `max` is only allowed in OData 4.0 responses; it is deprecated in OData 4.01. While clients MUST be prepared for this symbolic value, OData 4.01 and greater services MUST NOT return the symbolic value `max` and MAY instead specify the concrete maximum length supported for the type by the service, or omit the attribute entirely.

7.2.3 Precision

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

For a temporal ~~property the~~ value (datetime-with-offset, decimal, duration, or time-of~~ this attribute specifies~~-day): 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 28 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.

~~7.2.4~~ Attribute ~~Scale~~Precision

~~A decimal property MAY define~~The value of `Precision` is a number.

Example 17: `Precision` facet applied to the `DateTimeOffset` type

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

7.2.4 Scale

A non-negative integer value ~~or one of the symbolic values `floating` or `variable` for the `Scale` attribute.~~

~~This attribute specifies~~specifying the maximum number of digits allowed to the right of the decimal point, or one of the symbolic values `floating` or `variable`.

The value `floating` means that the decimal property represents a decimal floating-point number whose number of significant digits is the value of the ~~`Precision` attribute.~~`Precision` facet. OData 4.0 responses MUST NOT specify the value `floating`.

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~~`Precision` facet.

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~~facet minus the value of the `Scale` ~~attribute~~facet. If `Precision` is equal to `Scale`, a single zero ~~has to~~MUST precede the decimal point.

The value of ~~the~~ `Scale` ~~attribute~~ MUST be less than or equal to the value of ~~the `Precision` attribute.~~`Precision`. 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~~`Precision` 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 16: `Precision` and `Scale` facets applied to~~Attribute `Scale`

The value of `Scale` is a number or one of ~~the `Decimal` type~~symbolic values `floating` or `variable`.

Example 18: `Precision=3` and `Scale=2`.
Allowed values: 1.23, 0.23, 3.14 and 0.7, not allowed values: 123, 12.3~~.~~

```
<Property Name="~~Amount~~Amount32" Type="Edm.Decimal" Precision="3" Scale="2" />
```

Example 19~~: `Precision`~~: `Precision=2` equals ~~`Scale`.~~`Scale`.
Allowed values: 0.23, 0~~,~~ 7, not allowed values: 1.23, 1.2~~.~~

```
<Property Name="~~Amount~~Amount22" Type="Edm.Decimal" Precision="2" Scale="2" />
```

Example 20~~: `Precision` and a variable `Scale` applied to the `Decimal` type~~: `Precision=3` and a variable `Scale`.
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~~Amount3v" Type="Edm.Decimal" Precision="3"
Scale="variable" />
```

Example 21~~: `Precision` and a floating `Scale` applied to the `Decimal` type~~: `Precision=7` and a floating `Scale`.
Allowed values: -1.234567e3, 1e-101, 9.999999e96, not allowed would be: 1e-102 and 1e97 due to the limited precision.

```
<Property Name="~~Amount~~Amount3f" Type="Edm.Decimal" Precision="7"
Scale="floating" />
```


~~7.2.5 Attribute Unicode~~

7.2.5 ~~A~~Unicode

For a string property ~~MAY define a Boolean value for~~ the `Unicode` ~~attribute.~~

~~The value `true`~~facet indicates ~~that~~whether the property might contain and accept string values with Unicode characters beyond the ASCII character set. The value `false` indicates that the property will only contain and accept string values with characters limited to the ASCII character set.

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

~~7.2.6~~ Attribute ~~SRID~~Unicode

~~A~~The value of `Unicode` is one of the Boolean literals `true` or `false`. Absence of the attribute means `true`.

7.2.6 SRID

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

The value of the `SRID` ~~attribute~~facet 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~~facet and their meanings are as defined by the European Petroleum Survey Group **[EPSG]**.

~~7.2.7~~ Attribute ~~DefaultValue~~ SRID

The value of `$SRID` is a number or the symbolic value `variable`.

7.2.7 Default Value

A primitive or enumeration property MAY define a ~~value for the `DefaultValue` attribute. The value of this attribute determines the~~default value ~~of the property~~that is used 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.

Attribute DefaultValue

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.

8 Navigation Property

8 Navigation Property

~~8.1 Element edm:NavigationProperty~~

A navigation property allows navigation to related entities. It MUST specify a unique name as well as a type.

The navigation property's name MUST be a simple identifier. It is used when referencing, serializing or deserializing the navigation property. It MUST be unique within the set of structural and navigation properties of the declaring structured type, and MUST NOT match the name of any structural property in any of its base types. If a navigation property with the same name is defined in any of this type's base types, then the navigation property's type MUST be a type derived from the type specified for the navigation property of the base type, and constrains this navigation property to be of the specified subtype for instances of this structured type. The name MUST NOT match the name of any structural or navigation property of any of this type's base types for OData 4.0 responses.

Element edm:NavigationProperty

The `edm:NavigationProperty` element MUST contain the `Name` and `Type` attributes, and it MAY contain the attributes `Nullable`, `Partner`, and `ContainsTarget`.

It MAY contain child elements `edm:ReferentialConstraint` and at most one child element `edm:OnDelete`.

It MAY contain `edm:Annotation` elements.

Attribute Name

The value of `Name` is the navigation property's name.

Example 22: 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~~self.Category"
Nullable="false"
                      Partner="Products" />
  <NavigationProperty Name="Supplier" Type="~~Self~~self.Supplier" />
</EntityType>

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

8.1 Navigation Property Type

The navigation property's type MUST be an entity type in scope, the abstract type `Edm.EntityType`, or a collection of one of these types.

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

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

For a collection-valued containment navigation property the specified entity type MUST have a key defined.

A collection-valued navigation property may be annotated with the `Core.Ordered` term, defined in **[OData-CoreVoc]**, to specify that it supports a stable ordering.

A collection-valued navigation property may be annotated with the `Core.PositionalInsert` term, defined in **[OData-CoreVoc]**, to specify that it supports inserting items into a specific ordinal position.

~~8.1.1~~ Attribute ~~Name~~Type

~~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 defined in the containing structured type, and MUST NOT match the name of any structural property in any of its base types. If a navigation property with the same name is defined in any of this type's base types, then the value of the `Type` attribute of this navigation property MUST specify a type derived from the type specified for the navigation property of the base type, and constrains this navigation property to be of the specified subtype for instances of this structured type. The name MUST NOT match the name of any structural or navigation property of any of this type's base types for responses with an edmx `version` attribute of `4.0`.~~

~~8.1.2 Attribute `Type`~~

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

~~If the `ContainsTarget` attribute is `true`, and the navigation property is collection-valued, the specified entity type MUST have a key defined.~~

~~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.~~

~~8.1.3 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.~~

~~8.1.4 Attribute `Partner`~~

For single-valued navigation properties the value of `Type` is the qualified name of the navigation property's type.

For collection-valued navigation properties the value of `Type` is the character sequence `Collection(` followed by the qualified name of the navigation property's item type, followed by a closing parenthesis `)`.

8.2 Nullable Navigation Property

A Boolean value specifying whether the declaring type MAY have no related entity. If false, instances of the declaring structured type MUST always have a related entity.

Nullable MUST NOT be specified for a collection-valued navigation property, a collection is allowed to have zero items.

Attribute `Nullable`

The value of `Nullable` is one of the Boolean literals `true` or `false`. Absence of the attribute means `true`.

8.3 Partner Navigation Property

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

~~This attribute MUST NOT be specified for navigation~~. Navigation properties of complex types MUST NOT specify a partner.

If specified, the ~~value of this attribute MUST be~~partner navigation property is identified by a path ~~from~~relative to the entity type specified ~~in~~as the ~~`Type` attribute~~type of the navigation property. This path MUST lead to a navigation property defined on that type or a derived type. The path ~~may~~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~~declaring entity type of the current navigation property or one of its parent entity types.

If the ~~`Partner` attribute identifies a~~partner navigation property is single-valued ~~navigation property, the partner navigation property~~, it MUST lead back to the source entity from all related entities. If the ~~`Partner` attribute identifies a~~partner navigation property is collection-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.~~navigation property. The latter can occur if the partner navigation property is defined on a complex type, or if the current navigation property is defined on a type derived from the type of the partner navigation property.

~~8.1.5~~ Attribute ~~`ContainsTarget`~~`Partner`

The value of `Partner` is the path to the of the partner navigation property.

8.4 Containment Navigation Property

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~~indicate that instances of its declaring structured type contain the ~~value~~targets of the ~~`ContainsTarget` attribute is `true`,~~navigation property, in which case the navigation property is called a *containment navigation property*.

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

Instances of the structured 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 instance, followed by the path segment of the navigation property and the key of the contained entity, see **[OData-URL]**.

Entity types used in collection-valued containment navigation properties MUST have a key defined.

For items of an ordered collection of complex types (those annotated with the `Core.Ordered` term defined in **[OData-CoreVoc]**), the canonical URL of the item is the canonical URL of the collection appended with a segment containing the zero-based ordinal of the item. Items within in an unordered collection of complex types do not have a canonical URL. Services that support unordered collections of

complex types declaring a containment navigation property, either directly or indirectly via a property of complex type, MUST specify the URL for the navigation link within a payload representing that item, according to format-specific rules.

~~Responses with an edmx `version` attribute of `4.0`~~OData 4.0 responses MUST NOT specify a complex type declaring a containment navigation property 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.~~path 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~~partner navigation property. If the containment is recursive, the relationship defines a tree, thus the partner navigation property MUST be ~~nullable~~Nullable (for the root of the tree) and ~~specify a~~ single ~~entity type~~-valued (for the parent of a non-root entity). If the containment is not recursive, the partner navigation property MUST NOT be nullable.

An entity type inheritance chain MUST NOT contain more than one navigation property with a ~~`Partner` attribute referencing~~partner navigation property that is a containment ~~relationship~~navigation property.

Note: without a partner ~~attribute~~navigation property, 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.

~~8.2 Element `edm:ReferentialConstraint`~~

~~A~~Attribute `ContainsTarget`

The value of `ContainsTarget` is one of the Boolean literals `true` or `false`. Absence of the attribute means `false`.

8.5 Referential Constraint

A single-valued 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* declaring the navigation property) MUST have the same value as the *principal property* (the referenced property declared 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, or both types MUST be complex types.

If the principle property is an entity type, then the dependent property must reference the same entity.

If the principle property is a complex type, then the dependent property must reference a complex type with the same properties, each with the same values.

If the navigation property on which the referential constraint is defined is nullable, or the principal property is nullable, then the dependent property MUST also 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~~NOT be nullable.

~~*Example 21: 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">~~

~~<~~Element edm:ReferentialConstraint ~~Property="CategoryID"~~

The edm:ReferentialConstraint element MUST contain the attributes Property and ReferencedProperty~~="ID" />~~

~~</NavigationProperty>~~.

~~</EntityType>~~

It MAY contain edm:Annotation elements.

~~8.2.1~~ 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 property of the dependent entity type itself or to a 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.

~~8.2.2~~ 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 property of the principal entity type itself or to a 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.

~~8.3 Element edm:OnDelete~~

Example 23: the category must exist for a product in that category to exist. The CategoryID of the product is identical to the ID of the category, and the CategoryKind property of the product is identical to the Kind property of the category.

```
<EntityType Name="Product">
  …
  <Property Name="CategoryID" Type="Edm.String" Nullable="false"/>
  <Property Name="CategoryKind" Type="Edm.String" Nullable="true" />
  <NavigationProperty Name="Category" Type="self.Category" Nullable="false">
    <ReferentialConstraint Property="CategoryID" ReferencedProperty="ID" />
    <ReferentialConstraint Property="CategoryKind" ReferencedProperty="Kind">
      <Annotation Term="Core.Description"
                  String="Referential Constraint to non-key property" />
    </ReferentialConstraint>
  </NavigationProperty>
</EntityType>

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

8.6 On-Delete Action

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

~~*Example 22: 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>
```

~~8.3.1 Attribute `Action`~~

The ~~`edm:OnDelete` element MUST include the `Action` attribute with~~action can have 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~~ on-delete action is ~~present~~specified, the action taken by the service is not predictable by the client and could vary per entity.

Element `edm:OnDelete`

9 Complex Type

The `edm:OnDelete` element MUST contain the `Action` attribute.

It MAY contain `edm:Annotation` elements.

Attribute `Action`

The value of `Action` is one of the values `Cascade`, `None`, `SetNull`, or `SetDefault`.

Example 24: 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">
      <Annotation Term="Core.Description"
                  String="Delete all products in this category" />
    </OnDelete
  </NavigationProperty>
</EntityType>
```


9 Complex Type

Complex types are keyless ~~nominal~~nominal ~~structured types.~~structured types. The lack of a key means that instances of 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.

The complex type's name is a simple identifier that MUST be unique within its schema.

A complex type can define two types of properties. ~~A structural property~~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~~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.~~

Element `edm:ComplexType`

The `edm:ComplexType` element MUST contain the `Name` attribute, and it MAY contain the `BaseType`, `Abstract`, and `OpenType` attributes.

It MAY contain `edm:Property` and `edm:NavigationProperty` elements describing the properties of the complex type.

It MAY contain `edm:Annotation` elements.

Attribute `Name`

The value of `Name` is the complex type's name.

Example 25: 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~~self.Dimensions" />
  <Property Name="ShippingDimensions" Type="~~Self~~self.Dimensions" />
</EntityType>

<EntityType Name="ShipmentBox">
  ...…
  <Property Name="Dimensions" Type="~~Self~~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 Derived Complex Type

~~9.1.2 Attribute `BaseType`~~

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

A complex type inherits the structural and navigation properties ~~declared on the complex type's~~of its base type.

A complex type MUST NOT introduce an inheritance cycle ~~via the~~ by specifying a base type ~~attribute~~.

Attribute `BaseType`

~~9.1.3 Attribute `Abstract`~~

The value of `BaseType` is the qualified name of the base type.

9.2 Abstract Complex Type

A complex type MAY indicate that it is abstract and cannot ~~be instantiated by providing a Boolean value of `true` to the~~ have instances.

Attribute `Abstract` ~~attribute.~~

~~If not specified, the~~ The value of `Abstract` ~~attribute defaults to~~ is one of the Boolean literals `true` or `false`. Absence of the attribute means `false`.

9.3 Open Complex Type

~~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~~and allows clients to add properties dynamically to instances of the type by specifying uniquely named property 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~~indicate that it is also open.

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~~

~~Attribute `OpenType`~~

~~The value of `OpenType` is one of the Boolean literals `true` or `false`. Absence of the attribute means `false`.~~

10 Enumeration Type

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

The enumeration type's name is a simple identifier that MUST be unique within its schema.

Although enumeration types have an underlying numeric value, the preferred representation for an enumeration value is the member name. Discrete sets of numeric values should be represented as numeric values annotated with the `AllowedValues` annotation defined in **[OData-VocCore].**

~~The `IsFlags` attribute indicates~~Enumeration types marked as flags allow values that consist of more than one enumeration member ~~may be selected~~ at a time.

Element `edm:EnumType`

The `edm:EnumType` element MUST contain the Name attribute, and it MAY contain the `UnderlyingType` and `IsFlags` attributes.

It MAY contain one or more `edm:Member` elements defining the members of the enumeration type.

It MAY contain `edm:Annotation` elements.

Attribute `Name`

The value of `Name` is the enumeration type's name.

Example 26: 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 Underlying Integer Type

~~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`~~.~~ as its underlying type.

If ~~the `UnderlyingType` attribute is not~~not explicitly specified, `Edm.Int32` is used as the underlying type.

Attribute UnderlyingType

The value of UnderlyingType is the qualified name of the underlying type.

10.2 Flags Enumeration Type

~~10.1.3 Attribute IsFlags~~

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

If ~~no value is~~ not explicitly specified ~~for this attribute, its value defaults to~~ false.

~~10.2 Element edm:Member~~

~~The~~ edm:Member ~~element defines the discrete options for the~~, only one enumeration type member MAX be selected simultaneously.

Attribute IsFlags

~~*Example 25: 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~~IsFlags is one of the Boolean literals true or false. Absence of 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 26:* *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>
```

> attribute means false.

Example 27: 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~~

10.3 Enumeration Type Member

Enumeration type values consist of discrete members.

Each member is identified by its name, a simple identifier that MUST be unique within the enumeration type.

Each member MUST specify an associated numeric value that MUST be a valid value for the underlying type of the enumeration type.

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

Enumeration members are sorted by their numeric value.

~~11.1~~ Element edm:~~TypeDefinition~~Member

The `edm:Member` element MUST contain the `Name` attribute and it MAY contain the `Value` attribute.

It MAY contain `edm:Annotation` elements.

Attribute `Name`

The value of `Name` is the enumeration member's name.

Attribute `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.

Example 28: `FirstClass` has a value of `0`, `TwoDay` a value of 1, and `Overnight` a value of 2.

```
<EnumType Name="ShippingMethod">
  <Member Name="FirstClass">
    <Annotation Term="Core.Description"
                String="Shipped with highest priority" />
  </Member>
  <Member Name="TwoDay">
    <Annotation Term="Core.Description"
                String="Shipped within two days" />
  </Member>
  <Member Name="Overnight">
    <Annotation Term="Core.Description"
                String="Shipped overnight" />
  </Member>
</EnumType>
```


11 Type Definition

A type definition defines a specialization of one of the ~~primitive types~~primitive types.

The type definition's name is a simple identifier that MUST be unique within its schema.

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`~~

It is up to the definition of a term to specify whether and how annotations with this term propagate to places where the annotated type definition is used, and whether they can be overridden.

Element `edm:TypeDefinition`

The `edm:TypeDefinition` element MUST ~~provide the QualifiedName of a primitive type as~~contain the `Name` and `UnderlyingType` attributes.

It MAY contain `edm:Annotation` elements.

Attribute `Name`

The value of `Name` is the ~~`UnderlyingType` attribute. This type MUST NOT be another type definition.~~type definition's name.

~~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 29:

```
<TypeDefinition Name="Length" UnderlyingType="Edm.Int32">
  <Annotation Term="Org.OData.Measures.V1.Unit"
              String="Centimeters" />
</TypeDefinition>

<TypeDefinition Name="Weight" UnderlyingType="Edm.Int32">
  <Annotation Term="Org.OData.Measures.V1.Unit"
              String="Kilograms" />
</TypeDefinition>

<ComplexType Name="Size">
  <Property Name="Height" Type="~~Self~~self.Length" />
  <Property Name="Weight" Type="~~Self~~self.Weight" />
```



```
</ComplexType>
```

11.1 Underlying Primitive Type

The underlying type of a type definition MUST be a primitive type that MUST NOT be another type definition.

Attribute `UnderlyingType`

The value of `UnderlyingType` is the qualified name of the underlying type.

The type definition MAY specify facets applicable to the underlying type. Possible facets are: `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.

Where type definitions are used, the type definition is returned in place of the primitive type wherever the type is specified in a response.

~~```~~
~~</ComplexType>~~
~~```~~

12 Action and Function

12.1 Element edm:Action

The edm:Action element represents an action in an entity model.

12 Action and Function

12.1 Action

Actions are service-defined operations that 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's name is a simple identifier that MUST be unique within its schema.

Actions cannot be composed with additional path segments.

An action MAY specify a ~~return type using the `edm:ReturnType` element. The return type must~~return type that MUST be a primitive, entity or complex type, or a collection of primitive, entity or complex types in scope.

~~The~~An action ~~may also~~MAY define ~~zero or more `edm:Parameter` elements to be~~parameters 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~~

12.2 ~~Bound~~Action Overloads

Bound actions support overloading (multiple actions having the same name within the same ~~namespace~~schema) by binding parameter type. The combination of action name and the binding parameter type MUST be unique within a ~~namespace~~schema.

~~Unbound~~Unbound actions do not support overloads. The names of all unbound actions MUST be unique within a ~~namespace~~schema.

An unbound action MAY have the same name as a bound action.

> ### Element `edm:Action`
>
> The `edm:Action` element MUST contain the `Name` attribute and it MAY contain the `IsBound` and `EntitySetPath` attributes.
>
> It MAY contain at most one `edm:ReturnType` element and MAY contain `edm:Parameter` elements.
>
> It MAY contain `edm:Annotation` elements.
>
> #### ~~12.1.2~~ Attribute ~~`IsBound`~~`Name`
>
> The value of `Name` is the action's name.

12.3 Function

Functions are service-defined operations that MUST NOT have observable side effects and MUST return a single instance or a collection of instances of any type.

The function's name is a simple identifier that MUST be unique within its schema.

Functions MAY be composable.

The function MUST specify a return type which MUST be a primitive, entity or complex type, or a collection of primitive, entity or complex types in scope.

A function MAY define parameters to be used during the execution of the function.

12.4 Function Overloads

Bound functions support overloading (multiple functions having the same name within the same schema) 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 schema.
- The combination of function name, binding parameter type, and ordered set of parameter types MUST be unique within a schema.
- All bound functions with the same function name and binding parameter type within a schema 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 schema.
- The combination of function name and ordered set of parameter types MUST be unique within a schema.
- All unbound functions with the same function name within a schema MUST specify the same return type.

An unbound function MAY have the same name as a bound function.

~~An action element MAY specify a Boolean value for the `IsBound` attribute.~~

~~Actions whose `IsBound`~~Note that type definitions can be used to disambiguate overloads for both bound and unbound functions, even if they specify the same underlying type.

Element `edm:Function`

The `edm:Function` element MUST contain the `Name` attribute ~~is `false` or~~and it MAY contain the `IsBound` and `EntitySetPath` attributes.

It MUST contain one `edm:ReturnType` element, and it MAY contain `edm:Parameter` elements.

It MAY contain `edm:Annotation` elements.

Attribute `Name`

The value of `Name` is the action's name.

12.5 Bound or Unbound Action or Function Overloads

An action or function overload MAY indicate that it is bound. If not ~~specified are considered~~explicitly indicated, it is unbound.

Bound actions or functions are invoked on resources matching the type of the binding parameter. The binding parameter can be of any type, and it MAY be Nullable.

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~~ Unbound functions are invoked as static functions within a filter or orderby expression, or from the entity container through a function import.

Attribute `IsBound`

The value of `IsBound` is one of the Boolean literals `true` or `false`. Absence of the attribute means `false`.

12.6 Entity Set Path

Bound actions and functions that return an entity or a collection of entities MAY specify ~~a value for the `EntitySetPath` attribute~~an entity set path if ~~determination~~the entity set of the ~~entity set for the return type is contingent~~returned entities depends on the entity set of the binding parameter value.

~~The value for the `EntitySetPath` attribute~~The entity set path 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~~simple identifier of the ~~navigation property~~navigation property to be traversed. A type~~-~~cast segment names the ~~QualifiedName~~qualified name of the entity type that should be returned from the type cast.

Attribute `EntitySetPath`

~~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 of `EntitySetPath` 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~~ the entity set ~~of non-binding parameter names MUST be unique within a namespace.~~ path.
- ~~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.~~

12.7 Composable Function

~~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~~ MAY indicate that it is ~~`false` or~~ composable. If 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~~explicitly indicated, it 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~~not composable.

A composable function can be invoked with additional path segments or key predicates appended to the resource 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~~ Attribute `IsComposable`
>
> The value of `IsComposable` is one of the Boolean literals `true` or `false`. Absence of the attribute means `false`.

12.8 Return Type

The return type 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~~an action or function overload MAY be any type ~~casts.~~

~~A navigation segment names the SimpleIdentifier of the navigation property to be traversed. A~~in scope, or a collection of any type ~~cast segment names the QualifiedName of the entity type that should be returned from the type cast~~in scope.

12.3 Element edm:ReturnType

The attributes MaxLength, Precision, Scale, and SRIDThe facets Nullable, MaxLength, Precision, Scale, and SRID can be used to specify the facets of the return type, as appropriate to specify value restrictions of the return type, as well as the UnicodeUnicode facet for 4.01 and greater payloads. If the facet attributes are not specified, their values are considered unspecified.

Element edm:ReturnType

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

AThe edm:ReturnType element MUST contain the Type attribute, and it MAY contain the attributes Nullable, MaxLength, Unicode, Precision, Scale, and SRID.

It MAY contain edm:Annotation elements.

Attribute Type

For single-valued return type MAY specify a Boolean types the value forof Type is the Nullable attribute. If not specified, the Nullable attribute defaults to true.

Ifqualified name of the return type has a Type attribute that does not specify a .

For collection,-valued return types the value of Type is the character sequence Collection(followed by the qualified name of the return item type, followed by a closing parenthesis).

Attribute Nullable

The value of Nullable is one of the Boolean literals true or false. Absence of the attribute means true.

For single-valued return types the value true means that the action or function mayMAY return a single null value. AThe value of false means that the action or function will never return a null value and instead will fail with an error response if it cannot compute a result.

If the return type has a Type attribute that specifies a For collection,-valued return types the result will always exist, but thebe a collection that MAY be empty. In this case, the Nullable attribute applies to membersitems of the collection and specifies whether the collection canMAY contain null values.

12.9 Parameter

An action or function overload MAY specify parameters.

A bound action or function overload MUST specify at least one parameter; the first parameter is the binding parameter.

Each parameter MUST have a name that is a simple identifier. The parameter name MUST be unique within the action or function overload.

The parameter MUST specify a type. It MAY be any type in scope, or a collection of any type in scope.

The facets MaxLength, Precision, Scale, or SRID can be used as appropriate to specify value restrictions of the parameter, as well as the Unicode facet for 4.01 and greater payloads.

12.4 Element edm:Parameter

The edm:Parameter element allows one or more parameters to be passed to a function or action.

The `edm:Parameter` element MUST contain the `Name` and the `Type` attribute, and it MAY contain the attributes `Nullable`, `MaxLength`, `Unicode`, `Precision`, `Scale`, and `SRID`.

It MAY contain `edm:Annotation` elements.

Attribute `Name`

The value of `Name` is the parameter's name.

Attribute `Type`

For single-valued parameters the value of `Type` is the qualified name of the parameter.

For collection-valued parameters the value of `Type` is the character sequence `Collection(` followed by the qualified name of the return item type, followed by a closing parenthesis `)`.

Attribute `Nullable`

The value of `Nullable` is one of the Boolean literals `true` or `false`. Absence of the attribute means `true`.

The value `true` means that the parameter accepts a `null` value.

Example 30: 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~~self.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, as well as the `Unicode` facet for 4.01 and greater payloads. The descriptions of these facets and their implications are covered in section 7.2.~~

~~13 Entity Container~~

13 Entity Container

Each metadata document used to describe an OData service MUST define exactly one entity container.

The entity container's name is a simple identifier that MUST be unique within its schema.

Entity containers define the entity sets, singletons, function and action imports exposed by the service.

~~An *entity set*~~Entity set, singleton, action import, and function import names MUST be unique within an entity container.

An *entity set* allows access to entity type instances. Simple entity models frequently have one entity set per entity type.

Example 31: one entity set per entity type

```
<EntitySet Name="Products"   EntityType="~~Self~~self.Product" />
<EntitySet Name="Categories" EntityType="~~Self~~self.Category" />
```

Other entity models may expose multiple entity sets per type.

Example 32: three entity sets referring to the two entity types

```
<EntitySet Name="StandardCustomers" EntityType="~~Self~~self.Customer">
  <NavigationPropertyBinding Path="Orders" Target="Orders" />
</EntitySet>
<EntitySet Name="PreferredCustomers" EntityType="~~Self~~self.Customer">
  <NavigationPropertyBinding Path="Orders" Target="Orders" />
</EntitySet>
<EntitySet Name="Orders" EntityType="~~Self~~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~~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*~~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*~~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 32: function import returning the top ten revenue-generating products for a given fiscal year*~~

```
<FunctionImport Name="TopSellingProducts"
                Function="Model.TopSellingProducts"
                EntitySet="Products" />
```

Element `edm:EntityContainer`

The `edm:EntityContainer` MUST contain one or more `edm:EntitySet`, `edm:Singleton`, `edm:ActionImport`, or `edm:FunctionImport` elements.

It MAY contain `edm:Annotation` elements.

Attribute `Name`

The value of `Name` is the entity container's name.

Example 33: An entity container aggregates entity sets, singletons, action imports, and function imports.

```
<EntityContainer Name="DemoService">
  <EntitySet Name="Products" EntityType="~~Self~~self.Product">
```



```
    <NavigationPropertyBinding Path="Category" Target="Categories" />
    <NavigationPropertyBinding Path="Supplier" Target="Suppliers" />
  </EntitySet>
  <EntitySet Name="Categories" EntityType="Selfself.Category">
    <NavigationPropertyBinding Path="Products" Target="Products" />
  </EntitySet>
  </EntitySet>
  <EntitySet Name="Suppliers" EntityType="Selfself.Supplier">
    <NavigationPropertyBinding Path="Products" Target="Products" />
  </EntitySet>
  </EntitySet>
  <Singleton Name="MainSupplier" Type="Selfself.Supplier" />
  <ActionImport Name="LeaveRequestApproval" Action="Selfself.Approval" />
  <FunctionImport Name="ProductsByRating" Function="Selfself.ProductsByRating"
                  EntitySet="Products" />
</EntityContainer>
```

13.1 Element edm:EntityContainer

13.1 The edm:EntityContainer element represents anExtending an Entity Container

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 anspecify that it extends another 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.by extending entity containers. Clients should be prepared to process cycles introduced withby extending entity containers.

Attribute Extends

The value of Extends is the qualified name of the entity container to be extended.

Example 34: 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

13.2 The edm:EntitySet element representsEntity Set

Entity sets are top-level collection-valued resources.

An entity set is identified by its name, a simple identifier that MUST be unique within its entity container.

An entity set MUST specify a type that MUST be 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.~~scope.

An entity set MUST contain only instances of ~~the entity type~~ its specified ~~by the `EntityType` attribute~~entity type or its subtypes. The entity type ~~named by the `EntityType` attribute~~ MAY be ~~abstract~~abstract but MUST have a ~~key~~key defined.

~~13.2.3 Attribute `IncludeInServiceDocument`~~

~~The `edm:EntitySet` element~~An entity set MAY ~~include the `IncludeInServiceDocument` attribute whose Boolean value indicates~~indicate whether ~~the entity set~~it is ~~advertised~~included in the service document.

If ~~no value is specified for this attribute, its value defaults to `true`~~not explicitly indicatet, it is included.

Entity sets that cannot be queried without specifying additional query options SHOULD ~~specify the value `false` for this attribute~~NOT be included in the service document.

Element `edm:EntitySet`

~~13.3 Element `edm:Singleton`~~

The `edm:~~Singleton~~EntitySet` element MUST contain the attributes `Name` and `EntityType`, and it MAY contain the `IncludeInServiceDocument` attribute.

It MAY contain `edm:NavigationPropertyBinding` elements.

It MAY contain `edm:Annotation` elements.

Attribute `Name`

~~represents a single~~The value of `Name` is the entity set's name.

Attribute `EntityType`

The value of `EntityType` is the qualified name of an entity type in scope.

Attribute `IncludeInServiceDocument`

The value of `IncludeInServiceDocument` is one of the Boolean literals `true` or `false`. Absence of the attribute means `true`.

13.3 Singleton

Singletons are top-level single-valued resources.

A singleton is identified by its name, a simple identifier that MUST be unique within its entity container.

A singleton MUST specify a type that MUST be 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~~scope.

A singleton MUST reference an instance ~~of~~its entity type.

Element edm:Singleton

The edm:Singleton element MUST include the attributes Name and Type.

It MAY contain edm:NavigationPropertyBinding elements.

It MAY contain edm:Annotation elements.

Attribute Name

The value of Name is the singleton's name.

Attribute Type

The value of Type is whose value is the qualified name of an entity type in scope.

13.4 Navigation Property Binding

If the entity type ~~specified by the Type attribute~~of an entity set or singleton declares navigation properties, a navigation property binding allows describing which entity set or singleton will contain the related entities.

~~13.4 Element edm:NavigationPropertyBinding~~

An ~~entity set~~entity set or a ~~singleton~~singleton SHOULD contain ~~an edm:NavigationPropertyBinding element~~a navigation property binding for each ~~navigation property~~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~~

13.4.1 Binding Path

A navigation property binding MUST ~~name~~specify a path to a navigation property of the entity set's~~,~~ or singleton's declared entity type, or a navigation property reached through a chain of type casts, complex properties, or containment navigation ~~property's entity type or one of its subtypes in the Path attribute.~~properties. If the navigation property is defined on a subtype, the path ~~attribute~~ MUST contain the ~~QualifiedName~~qualified name 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 navigation property segment MUST be a non-containment navigation property and there MUST NOT be any non-containment navigation properties prior to the final navigation property segment.

OData 4.01 services MAY have a type-cast segment as the last path segment, allowing to bind instances of different sub-types to different targets.

The same navigation property path MUST NOT be specified 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. Note that it is possible to have navigation property bindings for paths that differ only in a type-cast segment, allowing to bind instances of different sub-types to different targets. If paths differ only in type-cast segments, the most specific path applies.

~~13.4.2 Attribute `Target`~~

13.4.2 Binding Target

A navigation property binding MUST specify a ~~SimpleIdentifier~~target via a simple identifier or ~~TargetPath value for the `Target` attribute that~~target path. It 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~~entities.

~~If the value of the `Target` attribute is a SimpleIdentifier~~If the target is a simple identifier, 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~~target is a ~~TargetPath~~target path, it MUST resolve to an entity set, singleton, or containment navigation property in scope. The path can traverse single-valued containment navigation properties or single-valued complex properties before ending in a containment navigation property, ~~but~~and there MUST ~~not~~NOT be any non-containment navigation properties prior to the final segment.

Element `edm:NavigationPropertyBinding`

The `edm:NavigationPropertyBinding` element MUST contain the attributes `Path` and `Target`.

Attribute `Path`

The value of `Path` is a path expression.

Attribute `Target`

The value of `Target` is a target path.

Example 35: for an entity set in the same container as the enclosing entity set `Categories`

```
<EntitySet Name="Categories" EntityType="~~Self~~self.Category">
  <NavigationPropertyBinding Path="Products"
                             Target="SomeSet" />
</EntitySet>
```

Example 36: for an entity set in any container in scope

```
<EntitySet Name="Categories" EntityType="~~Self~~self.Category">
  <NavigationPropertyBinding Path="Products"
                             Target="SomeModel.SomeContainer/SomeSet" />
</EntitySet>
```

Example 37: binding `Suppliers` on `Products` contained within `Categories`

```
<EntitySet Name="Categories" EntityType="~~Self~~self.Category">
  <NavigationPropertyBinding Path="Products/Supplier"
                             Target="Suppliers" />
</EntitySet>
```


13.5 Action Import

~~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 sets are top-level resources that are never ~~advertised~~included 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~~

An action import is identified by its name, a simple identifier that MUST be unique within its entity container.

An action import MUST specify the name of an unbound action in scope.

If the ~~return type of the action specified in the~~ `Action` ~~attribute is~~imported action returns an entity or a collection of entities, a ~~SimpleIdentifier~~simple identifier or ~~TargetPath~~target path value MAY be specified ~~for the~~ `EntitySet` ~~attribute that names~~to identify the entity set ~~to which~~that contains the returned entities ~~belong~~. If a ~~SimpleIdentifier~~simple identifier is specified, it MUST resolve to an entity set defined in the same entity container. ~~If a TargetPath~~If a target path is specified, it MUST resolve to an entity set in scope.

Element `edm:ActionImport`

~~If the return type is not an entity or a collection of entities, a value~~ The `edm:ActionImport` element MUST ~~NOT be defined for~~contain the attributes `Name` and `Action`, and it MAY contain the `EntitySet` attribute.

~~13.6~~ Element ~~edm:FunctionImport~~

It MAY contain `edm:Annotation` elements.

Attribute `Name`

~~The~~ `edm:FunctionImport` ~~element allows exposing an unbound function as a~~The value of `Name` is the action import's name.

Attribute `Action`

The value of `Action` is the qualified name of an unbound action.

Attribute `EntitySet`

The value of `$EntitySet` is either the unqualified name of an entity set in the same entity container or a path to an entity set in a different entity container.

13.6 Function Import

Function imports sets are top-level ~~element in an~~ resources.

A function import is identified by its name, a simple identifier that MUST be unique within its entity container.

A function import MUST specify the name of an unbound function in scope. All unbound ~~overloads~~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~~imported function ~~specified in the `Function` attribute is~~returns an entity or a collection of entities, a ~~SimpleIdentifier~~simple identifier or ~~TargetPath~~target path value MAY be ~~defined for the `EntitySet` attribute that names~~specified to identify the entity set ~~to which~~that contains the returned entities ~~belong. If a SimpleIdentifier~~. If a simple identifier is specified, it MUST resolve to an entity set defined in the same entity container. ~~If a TargetPath~~If a target path is specified, it MUST resolve to an entity set in scope.

~~If the return type~~A function import for a parameterless function MAY indicate whether it is included in the service document. If not explicitly indicated, it is not included.

Element `edm:FunctionImport`

The `edm:FunctionImport` element MUST contain the attributes `Name` and `Function`, and it MAY contain the attributes `EntitySet` and `IncludeInServiceDocument`.

Attribute `Name`

~~an entity or a collection~~ The value of `Name` is the function import's name.

Attribute `Function`

~~entities, a~~The value ~~MUST NOT be defined for~~of `Function` is the qualified name of an unbound function.

Attribute `EntitySet`

The value of `EntitySet` ~~attribute~~is either the unqualified name of an entity set in the same entity container or a path to an entity set in a different entity container.

~~13.6.4~~ Attribute `IncludeInServiceDocument`

The ~~`edm:FunctionImport` for a parameterless function MAY include the~~ value of `IncludeInServiceDocument` is one of the Boolean literals `true` or `false`. Absence of the attribute ~~whose Boolean value indicates whether the function import is advertised in the service document~~means `false`.

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

~~14 Vocabulary and Annotation~~

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*~~ 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-JSON]**. Annotations that apply across instances should be specified as metadata annotations.

A *vocabulary* is a ~~namespace~~schema containing a set of terms where each ~~term~~ 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~~ term can be used to:

- ~~To extend~~Extend model elements and type instances with additional information.
- ~~To map~~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 38: 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.~~

14.1 Term

A term allows annotating a CSDL element or OData resource representation with additional data.

~~14.1.1 Attribute Name~~

The term's name is a simple identifier that MUST be unique within its schema.

The term's type MUST be a type in scope, or a collection of a type in scope.

Element edm:Term

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.~~contain the attributes Name and Type. It ~~indicates what type of value must be returned by the expression contained in an annotation using~~MAY contain the ~~term~~attributes BaseTerm and AppliesTo.

~~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.~~

It MAY specify values for the Nullable, MaxLength, Precision, Scale, or SRID facet attributes, as well as the Unicode facet attribute for 4.01 and greater payloads. These facets and their implications are described in section 7.2.

~~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.

It MAY contain edm:Annotation elements.

Attribute Name

The value of Name is the term's name.

Attribute Type

For single-valued properties the value of Type is the qualified name of the property's type.

For collection-valued properties the value of Type is the character sequence Collection(followed by the qualified name of the property's item type, followed by a closing parenthesis).

Attribute DefaultValue

The value of this attribute determines the value of the term when applied in an ~~edm:Annotation~~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`.

14.1.1 Specialized Term

~~14.1.5~~ A term ~~Attribute `AppliesTo`~~

~~The `edm:Term` element~~ MAY specialize another term in scope by specifying it as its base type.

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.

Attribute `BaseTerm`

~~define a value for the `AppliesTo` attribute.~~ The value of ~~this attribute is a whitespace-separated~~`BaseTerm` is the qualified name of the base term.

14.1.2 Applicability

A term MAY specify a list of ~~CSDL element names, or the value `Collection` indicating an element representing a collection, that this term~~ model elements it is intended to be applied to. If no ~~value~~list 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 term to be applied to any model element and SHOULD be prepared to handle unknown values within the `AppliesTo` attribute.

Symbolic Value	Model Element
`Action`	Action
`ActionImport`	Action Import
`Annotation`	Annotation
`Apply`	Application of a client-side function in an annotation
`Cast`	Type Cast annotation expression
`Collection`	Entity Set or collection-valued Property or Navigation Property
`ComplexType`	Complex Type
`EntityContainer`	Entity Container
`EntitySet`	Entity Set
`EntityType`	Entity Type
`EnumType`	Enumeration Type
`Function`	Function
`FunctionImport`	Function Import
`If`	Conditional annotation expression
`Include`	Reference to an Included Schema
`IsOf`	Type Check annotation expression

LabeledElement	Labeled Element expression
Member	Enumeration Member
NavigationProperty	Navigation Property
Null	Null annotation expression
OnDelete	On-Delete Action of a navigation property
Parameter	Action of Function Parameter
Property	Property of a structured type
PropertyValue	Property value of a Record annotation expression
Record	Record annotation expression
Reference	Reference to another CSDL document
ReferentialConstraint	Referential Constraint of a navigation property
ReturnType	Return Type of an Action or Function
Schema	Schema
Singleton	Singleton
Term	Term
TypeDefinition	Type Definition
UrlRef	UrlRef annotation expression

Attribute AppliesTo

The value of AppliesTo is a whitespace-separated list of symbolic values from the table above that identify model elements the term is intended to be applied to.

*Example 39: 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="IsURL" Type="Core.Tag" DefaultValue="true"
      AppliesTo="Property Term">
  <Annotation Term="Core.Description">
    <String>
      Properties and terms annotated with this term MUST contain a valid URL
    </String>
  </Annotation>
  <Annotation Term="Core.RequiresType" String="Edm.String" />
</Term>
```

14.2 Annotation

An annotation applies a term to a model element and defines how to calculate a value for the term application. Both term and model element MUST be in scope. Section 14.1.2 specifies which model elements MAY be annotated with a term.

The value of an annotation is specified as an *annotation expression*, which is either a constant expression representing a constant value, or a dynamic expression. The most common construct for assigning an annotation value is a path expression that refers to a property of the same or a related structured type.

Example 40: term `Measures.ISOCurrency`, once applied with a constant value, once with a path value

```
<Property Name="AmountInReportingCurrency" Type="Edm.Decimal">
  <Annotation Term="Measures.ISOCurrency" String="USD" />
</Property>
<Property Name="AmountInTransactionCurrency" Type="Edm.Decimal">
  <Annotation Term="Measures.ISOCurrency" Path="Currency" />
</Property>
<Property Name="Currency" Type="Edm.String" MaxLength="3" />
```

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.

Structured types “inherit” annotations from their direct or indirect base types. If both the type and one of its base types is annotated with the same term and qualifier, the annotation on the type completely replaces the annotation on the base type; structured or collection-valued annotation values are not merged. Similarly, properties of a structured type inherit annotations from identically named properties of a base type.

It is up to the definition of a term to specify whether and how annotations with this term propagate to places where the annotated model element is used, and whether they can be overridden. E.g. a "Label" annotation for a UI can propagate from a type definition to all properties using that type definition and may be overridden at each property with a more specific label, whereas an annotation marking a type definition as containing a phone number will propagate to all using properties but may not be overridden.

~~14.1.6 Term Facets~~

~~The `edm:Term` element MAY specify values for the `Nullable`, `MaxLength`, `Precision`, `Scale`, or `SRID` attributes, as well as `Unicode` for 4.01 and greater payloads. These facets and their implications are described in section 7.2.~~

~~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`~~

14.2.1 ~~The `edm:Annotations`~~Qualifier

A term can be applied multiple times to the same model element by providing a qualifier to distinguish the annotations. The qualifier is a simple identifier.

The combination of target model element, term, and qualifier uniquely identifies an annotation.

Element `edm:Annotation`

The `edm:Annotation` element MUST ~~include a `Target`~~contain the attribute ~~whose value is a path expression that MUST resolve to a model element in the entity model~~`Term`, and it MAY contain the attribute `Qualifier`.

The value of the annotation MAY be a constant expression or dynamic expression.

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 its properties. If no expression is specified for a collection-valued term, the annotation evaluates to an empty collection.

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.

Attribute `Term`

The value of `Term` is the qualified name of a term in scope.

Attribute `Qualifier`

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.

Example 41: annotation should only be applied to tablet devices

```
<Annotation Term="org.example.display.DisplayName" Path="FirstName"
            Qualifier="Tablet" />
```

14.2.2 ~~External~~Target

The target of an annotation is the model element the term is applied to.

The target of an annotation MAY be specified indirectly by "nesting" the annotation within the model element. Whether and how this is possible is described per model element in this specification.

The target of an annotation MAY also be specified directly; this allows defining an annotation in a different schema than the targeted model element.

This external targeting is only possible for ~~EDM~~model elements that are uniquely identified within their parent, and all their ancestor elements are uniquely identified within their parent:

- ~~`edm:Action`~~Action (applies to all overloads)
- ~~`edm:ActionImport`~~
- ~~`edm:Annotation`~~
- ~~`edm:ComplexType`~~
- ~~`edm:EntityContainer`~~
- ~~`edm:EntitySet`~~
- ~~`edm:EntityType`~~
- ~~`edm:EnumType`~~
- ~~`edm:Function`~~Action Import
- Annotation
- Complex Type
- Entity Container
- Entity Set
- Entity Type
- Enumeration Type
- Enumeration Type Member
- Singleton
- Type Definition
- Function (applies to all overloads)
- ~~`edm:FunctionImport`~~
- ~~`edm:Member`~~
- ~~`edm:NavigationProperty`~~Function Import
- Navigation Property (via type, entity set, or singleton)
- ~~`edm:Parameter`~~Parameter of an action or function (applies to all overloads defining the parameter)
- ~~`edm:Property`~~Property (via type, entity set, or singleton)
- ~~`edm:ReturnType`~~Return Type of an action or function (applies to all overloads)
- ~~`edm:Singleton`~~
- ~~`edm:Term`~~
- ~~`edm:TypeDefinition`~~

These are the direct children of a schema with a unique name (i.e. except actions and functions whose overloads to not possess a natural identifier), and all direct children of an entity container. ~~The `edm:Schema` element and most of the not uniquely identifiable EDM elements can still be annotated using an inline `edm:Annotation` element.~~

External targeting is possible for actions, functions, their parameters, and their return type, 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~~qualified name of schema child
- ~~QualifiedName~~qualified name of schema child followed by a forward slash and name of child element
- ~~QualifiedName~~qualified name of structured type followed by zero or more property, navigation property, or type~~-~~cast segments, each segment starting with a forward slash
- ~~QualifiedName~~qualified name 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
- ~~QualifiedName~~qualified name of an action or function followed by a forward slash and `$ReturnType`
- ~~QualifiedName~~qualified name of an entity container followed by a segment containing an action or function~~-~~ import name, optionally followed by a forward slash and either a parameter name or `$ReturnType`
- One of the preceding, followed by a forward slash, an at (`@`), the ~~QualifiedName~~qualified name of a term, and optionally a hash (`#`) and the qualifier of an annotation

Example 42: 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 41: 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 its properties. 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.

Structured types "inherit" annotations from their direct or indirect base types. If both the type and one of its base types is annotated with the same term and qualifier, the annotation on the type completely replaces the annotation on the base type; structured or collection-valued annotation values are not merged. Similarly, properties of a structured type inherit annotations from identically named properties of a base type.

It is up to the definition of a term to specify whether and how annotations with this term propagate to places where the annotated model element is used, and whether they can be overridden. E.g. a "Label" annotation for a UI can propagate from a type definition to all properties using that type definition and may be overridden at each property with a more specific label, whereas an annotation marking a type definition as containing a phone number will propagate to all using properties but may not be overridden.

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 42: 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

14.3 Constant Expression

Constant expressions allow assigning a constant value to an applied term. The constant expressions support element and attribute notation.

Example 43: 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>
```

14.3.1 Binary

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 43: base64url-encoded binary value (OData)

```
<Annotation Term="org.example.display.Thumbnail" Binary="T0RhdGE" />

<Annotation Term="org.example.display.Thumbnail">
  <Binary>T0RhdGE</Binary>
</Annotation>
```

14.3.2 Boolean

Expression edm:Bool

14.4.2 Expression edm:Bool

The edm:Bool expression evaluates to a primitive Boolean Boolean value. A Boolean expression MUST be assigned a BooleanBoolean value.

The BooleanBoolean expression MAY be provided using element notation or attribute notation.

Example 44: Example 45:

```
<Annotation Term="org.example.display.ReadOnly" Bool="true" />

<Annotation Term="org.example.display.ReadOnly">
  <Bool>true</Bool>
</Annotation>
```

14.3.3 Date

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 45:

```
<Annotation Term="org.example.vCard.birthDay" Date="2000-01-01" />

<Annotation Term="org.example.vCard.birthDay">
  <Date>2000-01-01</Date>
</Annotation>
```


14.3.4 DateTimeOffset

~~14.4.4~~ Expression `edm:DateTimeOffset`

The `edm:DateTimeOffset` expression evaluates to a primitive ~~date/time~~datetimestamp value with a time-zone offset. A ~~date/time~~datetimestamp 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~~datetimestamp expression MAY be provided using element notation or attribute notation.

Example 46:

```
<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>
```

14.3.5 Decimal

~~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 47~~:~~: attribute notation

```
<Annotation Term="org.example.display.Width" Decimal="3.14" />
```

Example 48: element notation

```
<Annotation Term="org.example.display.Width">
  <Decimal>3.14</Decimal>
</Annotation>
```

14.3.6 Duration

~~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 49:

```
<Annotation Term="org.example.task.duration" Duration="P7D" />

<Annotation Term="org.example.task.duration">
  <Duration>P11DT23H59M59.999999999999S</Duration>
</Annotation>
```


14.3.7 Enumeration Member

~~14.4.7~~ Expression `edm:EnumMember`

The `edm:EnumMember` expression references a ~~member~~member of an ~~enumeration type~~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 50: 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 51: 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>
```

14.3.8 Floating-Point Number

~~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 52:

```
<Annotation Term="org.example.display.Width" Float="3.14" />

<Annotation Term="org.example.display.Width">
  <Float>3.14</Float>
</Annotation>
```

14.3.9 Guid

~~14.4.9~~ Expression `edm:Guid`

The `edm:Guid` expression evaluates to a primitive ~~32-character string~~guid 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 53:

```
<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>
```

14.3.10 Integer

~~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 54: attribute notation

```
<Annotation Term="org.example.display.Width" Int="42" />
```

Example 55: element notation

```
<Annotation Term="org.example.display.Width">
  <Int>42</Int>
</Annotation>
```

14.3.11 String

~~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 56:

```
<Annotation Term="org.example.display.DisplayName" String="Product Catalog" />

<Annotation Term="org.example.display.DisplayName">
  <String>Product Catalog</String>
</Annotation>
```

14.3.12 Time of Day

~~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 57:

```
<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~~

14.4 Dynamic Expression

Dynamic expressions allow assigning a calculated value to an applied term. ~~The dynamic~~

14.4.1 Path Expressions

Path expressions ~~edm:AnnotationPath, edm:NavigationPropertyPath, edm:Path, edm:PropertyPath, and edm:UrlRef~~ allow assigning a value to an applied term or term component. There are two kinds of path expressions ~~support~~:

- A *model path* is used within Annotation Path, Model Element Path, Navigation Property Path, and Property Path expressions to traverse the model of a service and resolves to the model element ~~and attribute notation, all~~identified by the path. It allows assigning values to terms or term properties of the built-in types `Edm.AnnotationPath`, `Edm.NavigationPropertyPath`, `Edm.PropertyPath`, and their base types `Edm.AnyPropertyPath` and `Edm.ModelElementPath`.
- An *instance path* is used within a Value Path expression to traverse a graph of type instances and resolves to the value identified by the path. It allows assigning values to terms or term properties of built-in types other ~~dynamic expressions only support~~than the `Edm.*Path` types, or of any model-defined type.

14.4.1.1 Path Syntax

Model paths and instance paths share a common syntax which is derived from the path expression syntax of URLs, see **[OData-URL]**.

A path MUST be composed of zero or more path segments joined together by forward slashes (`/`).

Paths starting with a forward slash (`/`) are absolute paths, and the first path segment MUST be the qualified name of a model element ~~notation~~, e.g. an entity container. The remaining path after the second forward slash is interpreted relative to that model element.

~~14.5.1 Comparison and Logical Operators~~

~~*The following EDM elements allow service authors*~~*Example 58: absolute path to ~~supply~~an entity set*

```
/self.MyEntityContainer/MyEntitySet
```

Paths not starting with a ~~dynamic conditional~~forward slash are interpreted relative to the annotation target, following the rules specified in section “Path Evaluation”.

Example 59: relative path to a property

```
Address/City
```

If a path segment is a qualified name, it represents a *type cast*, and the segment MUST be the name of a type in scope. If the type or instance identified by the preceding path part cannot be cast to the specified type, the path expression ~~which~~ evaluates to ~~a~~the null value.

Example 60: type-cast segment

```
…/self.Manager/…
```

If a path segment starts with an at (`@`) character, it represents a *term cast*. ~~of type `Edm.Boolean`. They~~ The at (`@`) character MUST be followed by a qualified name that MAY be ~~combined~~followed by a hash (`#`) character and a simple identifier. The qualified name preceding the hash character MUST resolve to a term that is in scope, the simple identifier following the hash sign is interpreted as a qualifier for the term. If the model element or instance identified by the preceding path part has not been annotated with that

term (and if present, with that qualifier), the term cast evaluates to the null value. Three special terms are implicitly "annotated" for media entities and stream properties:

- `odata.mediaEditLink`
- `odata.mediaReadLink`
- `odata.mediaContentType`

Example 61: term-cast segment

```
…/@Capabilities.SortRestrictions/…
```

If a path segment is a simple identifier, it MUST be the name of a child model element of the model element identified by the preceding path part, or a structural or navigation property of the instance identified by the preceding path part.

A model path MAY contain any number of segments representing collection-valued structural or navigation properties. The result of the expression is the model element reached via this path.

Example 62: property segment in model path

```
…/Orders/Items/Product/…
```

An instance path MUST NOT contain more than one segment representing a collection-valued construct, e.g. an entity set or a collection-valued structural or navigation property. The result of the expression is the collection of instances resulting from applying any remaining segments that operate on a single-valued expression to each instance in the collection-valued segment.

An instance path MAY terminate in a `$count` segment if the previous segment is collection-valued, in which case the path evaluates to the number of items in the collection identified by the preceding segment.

Example 63: property segments in instance path

```
…/Addresses/Street
…/Addresses/$count
```

A model path MAY contain path segments starting with a navigation property, then followed by an at (`@`) character, then followed by the qualified name of a term in scope, and optionally followed by a hash (`#`) character and a simple identifier which is interpreted as a qualifier for the term. If the navigation property has not been annotated with that term (and if present, with that qualifier), the path segment evaluates to the null value. This allows addressing annotations on the navigation property itself; annotations on the entity type specified by the navigation property are addressed via a term-cast segment.

Example 64: model path addressing an annotation on a navigation property

```
…/Items@Capabilities.InsertRestrictions/Insertable
```

An instance path MAY contain path segments starting with an entity set or a collection-valued navigation property, then followed by a key predicate using parentheses-style convention, see **[OData-URL]**. The key values are either primitive literals or instance paths. If the key value is a relative instance path, it is interpreted according to the same rule below as the instance path it is part of, *not* relative to the instance identified by the preceding path part.

Example 65: instance path with entity set and key segment

```
/self.container/SettingsCollection('FeatureXxx')/IsAvailable
/self.container/Products(ID=ProductID)/Name
```

14.4.1.2 Path Evaluation

Annotations MAY be embedded within their target, or specified separately, e.g. as part of a different schema, and specify a path to their target model element. 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 is evaluated starting at the entity container, i.e. an empty path resolves to the entity container, and non-empty paths MUST start with a segment identifying 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 is evaluated starting at the entity set or singleton, i.e. an empty path resolves to the entity set or singleton, 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 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 structural or navigation property of the type, a type cast, or a term cast.

For annotations embedded within a structural or navigation property of an entity type or complex type, the path 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.

~~they MAY be used anywhere instead of an `edm:Bool` expression.~~

~~Element~~	~~Description~~	~~Example~~
~~**Logical Operators**~~		
~~`edm:And`~~	~~Logical and~~	~~`<And><Path>IsMale</Path><Path>IsMarried</Path></And>`~~
~~`edm:Or`~~	~~Logical or~~	~~`<Or><Path>IsMale</Path><Path>IsMarried</Path></Or>`~~
~~`edm:Not`~~	~~Logical negation~~	~~`<Not><Path>IsMale</Path></Not>`~~
~~**Comparison Operators**~~		
~~`edm:Eq`~~	~~Equal~~	~~`<Eq><Null/><Path>IsMale</Path></Eq>`~~
~~`edm:Ne`~~	~~Not equal~~	~~`<Ne><Null/><Path>IsMale</Path></Ne>`~~
~~`edm:Gt`~~	~~Greater than~~	~~`<Gt><Path>Price</Path><Int>20</Int></Gt>`~~
~~`edm:Ge`~~	~~Greater than or equal~~	~~`<Ge><Path>Price</Path><Int>10</Int></Ge>`~~
~~`edm:Lt`~~	~~Less than~~	~~`<Lt><Path>Price</Path><Int>20</Int></Lt>`~~
~~`edm:Le`~~	~~Less than or equal~~	~~`<Le><Path>Price</Path><Int>100</Int></Le>`~~

~~The `edm:And` and `edm:Or` elements require two child expressions that evaluate to Boolean values. The `edm:Not` element 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.~~

For annotations targeting a structural or navigation property of an entity type or complex type, the path is evaluated starting at the *outermost* entity type or complex type named in the target of the annotation, i.e. an empty path resolves to the outermost type, and the first segment of a non-empty path MUST be a structural or navigation property of the outermost type, a type cast, or a term cast.

For annotations embedded within or targeting an action, action import, function, or function import, the first segment of the path MUST be a parameter name or `$ReturnType`.

14.4.1.3 Annotation Path

~~14.5.2 Expression edm:AnnotationPath~~

The ~~edm:AnnotationPath~~ annotation path expression provides a value for terms or term properties that specify the ~~built-in abstract types~~built-in types `Edm.AnnotationPath` or `Edm.~~AnyPath~~` ~~. It uses the same syntax and rules as the~~`ModelElementPath`. Its argument is a ~~edm:Path expression,~~model path with the following ~~exceptions~~restriction:

- ~~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`~~The value of the annotation path 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.

Expression edm:AnnotationPath

The `edm:AnnotationPath` expression MAY be provided using element notation or attribute notation.

Example 66:

~~*Example 57:*~~

```
<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>
```

14.4.1.4 Model Element Path

The model element path expression provides a value for terms or term properties that specify the built-in type `Edm.ModelElementPath`. Its argument is a model path.

The value of the model element path expression is the path itself, not the instance(s) identified by the path.

~~14.5.3~~ Expression edm:~~Apply~~ModelElementPath

The `edm:ModelElementPath` expression MAY be provided using element notation or attribute notation.

Example 67:

```
<Annotation Term="org.example.MyFavoriteModelElement"
            ModelElementPath="/org.example.someAction" />

<Annotation Term="org.example.MyFavoriteModelElement">
  <ModelElementPath>/org.example.someAction</ModelElementPath>
</Annotation>
```


14.4.1.5 Navigation Property Path

The navigation property path expression provides a value for terms or term properties that specify the built-in types `Edm.NavigationPropertyPath`, `Edm.AnyPropertyPath`, or `Edm.ModelElementPath`. Its argument is a model path with the following restriction:

- The last path segment MUST resolve to either a navigation property, 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.

The value of the navigation property path expression is the path itself, not the instance(s) identified by the path.

Expression `edm:NavigationPropertyPath`

The `edm:NavigationPropertyPath` expression MAY be provided using element notation or attribute notation.

Example 68:

```
<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>
```

14.4.1.6 Property Path

The property path expression provides a value for terms or term properties that specify one of the built-in types `Edm.PropertyPath`, `Edm.AnyPropertyPath`, or `Edm.ModelElementPath`. Its argument is a model path with the following restriction:

- The last path segment MUST resolve either to a structural property, 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.

The value of the property path expression is the path itself, not the value of the property or the value of the term cast identified by the path.

Expression `edm:PropertyPath`

The `edm:PropertyPath` MAY be provided using either element notation or attribute notation.

Example 69:

```
<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>
```

14.4.1.7 Value Path

The value path expression allows assigning a value 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. Its argument is an instance path.

The value of the path expression is the instance or collection of instances identified by the path.

Expression `edm:Path`

The `edm:Path` expression MAY be provided using element notation or attribute notation.

Example 70:

```
<Annotation Term="org.example.display.DisplayName" Path="FirstName" />

<Annotation Term="org.example.display.DisplayName">
  <Path>@vCard.Address#work/FullName</Path>
</Annotation>
```

14.4.2 Comparison and Logical Operators

Annotations MAY use the following logical and comparison expressions which evaluate to a Boolean value. These expressions MAY be combined and they MAY be used anywhere instead of a Boolean expression.

Operator	Description
Logical Operators	
`And`	Logical and
`Or`	Logical or
`Not`	Logical negation
Comparison Operators	
`Eq`	Equal
`Ne`	Not equal
`Gt`	Greater than
`Ge`	Greater than or equal
`Lt`	Less than
`Le`	Less than or equal
`Has`	Has enumeration flag(s) set
`In`	Is in collection

The `And` and `Or` operators require two operand expressions that evaluate to Boolean values. The `Not` operator requires a single operand expression that evaluates to a Boolean value. For details on null handling for comparison operators see **[OData-URL]**.

The other comparison operators require two operand expressions that evaluate to comparable values.

Expressions edm:And and edm:Or

The And and Or logical expressions are represented as elements edm:And and edm:Or that MUST contain two annotation expressions.

It MAY contain edm:Annotation elements.

Expression edm:Not

Negation expressions are represented as an element edm:Not that MUST contain a single annotation expression.

It MAY contain edm:Annotation elements.

Expressions edm:Eq, edm:Ne, edm:Gt, edm:Ge, edm:Lt, edm:Le, edm:Has, and edm:In

All comparison expressions are represented as an element that MUST contain two annotation expressions.

They MAY contain edm:Annotation elements.

Example 71:

```
<And>
  <Path>IsMale</Path>
  <Path>IsMarried</Path>
</And>
<Or>
  <Path>IsMale</Path>
  <Path>IsMarried</Path>
</Or>
<Not>
  <Path>IsMale</Path>
</Not>
<Eq>
  <Null />
  <Path>IsMale</Path>
</Eq>
<Ne>
  <Null />
  <Path>IsMale</Path>
</Ne>
<Gt>
  <Path>Price</Path>
  <Int>20</Int>
</Gt>
<Ge>
  <Path>Price</Path>
  <Int>10</Int>
</Ge>
<Lt>
  <Path>Price</Path>
  <Int>20</Int>
</Lt>
<Le>
  <Path>Price</Path>
  <Int>100</Int>
</Le>
<Has>
  <Path>Fabric</Path>
  <EnumMember>org.example.Pattern/Red</EnumMember>
</Has>
<In>
  <Path>Size</Path>
```



```
  <Collection>
    <String>XS</String>
    <String>S</String>
  </Collection>
</In>
```

14.4.3 Arithmetic Operators

Annotations MAY use the following arithmetic expressions which evaluate to a numeric value. These expressions MAY be combined and they MAY be used anywhere instead of a numeric expression of the appropriate type. The semantics and evaluation rules for each arithmetic expression is identical to the corresponding arithmetic operator defined in **[OData-URL]**.

Operator	Description
`Add`	Addition
`Sub`	Subtraction
`Neg`	Negation
`Mul`	Multiplication
`Div`	Division (with integer result for integer operands)
`DivBy`	Division (with fractional result also for integer operands)
`Mod`	Modulo

The `Neg` operator requires a single operand expression that evaluates to a numeric value. The other arithmetic operators require two operand expressions that evaluate to numeric values.

Expression `edm:Neg`

Negation expressions are represented as an element `edm:Neg` that MUST contain a single annotation expression.

It MAY contain `edm:Annotation` elements.

Expressions `edm:Add`, `edm:Sub`, `edm:Mul`, `edm:Div`, `edm:DivBy`, and `edm:Mod`

These arithmetic expressions are represented as an element that MUST contain two annotation expressions.

They MAY contain `edm:Annotation` elements.

Example 72:

```
~~Apply~~<Add>
  <Path>StartDate</Path>
  <Path>Duration</Path>
</Add>
<Sub>
  <Path>Revenue</Path>
  <Path>Cost</Path>
</Sub>
<Neg>
  <Path>Height</Path>
</Neg>
<Mul>
  <Path>NetPrice</Path>
  <Path>TaxRate</Path>
</Mul>
```



```
<Div>
  <Path>Quantity</Path>
  <Path>QuantityPerParcel</Path>
</Div>
<DivBy>
  <Path>Quantity</Path>
  <Path>QuantityPerParcel</Path>
</DivBy>
<Mod>
  <Path>Quantity</Path>
  <Path>QuantityPerParcel</Path>
</Mod>
```

14.4.4 Apply Client-Side Function

The apply expression enables a value to be obtained by applying a client-side function. The ~~Apply~~apply expression MUST ~~contain~~have at least one operand expression. The operand expressions ~~contained within the Apply expression~~ are used as parameters to the function.

Expression edm: Apply

The edm:Apply element MUST contain the Function attribute and MUST contain at least one operand expression ~~MUST be written with element notation~~.

It MAY contain more edm:Annotation elements.

~~14.5.3.1~~ Attribute Function

The ~~edm:Apply expression MUST include a~~value of Function ~~attribute whose value is a QualifiedName specifying~~ the qualified 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~~

14.4.4.1 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 73:

```
<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~~14.4.4.2 ~~Function `odata.fillUriTemplate`~~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~~labeled element expressions. Each labeled element 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~~labeled element 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~~labeled element expressions that evaluate to a collection of primitive values.

Key-value maps are represented as ~~`edm:LabeledElement` expressions~~labeled element 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 74: assuming there are no special characters in values of the `Name` property of the `Actor` entity

```
<Apply Function="odata.fillUriTemplate">
  <String>http://host/someAPI/Actors/{actorName}/CV</String>
  <LabeledElement Name="actorName" Path="Actor/Name" />
</Apply>
```

14.4.4.3 ~~Function `odata.uriEncode`~~Function `odata.matchesPattern`

The `odata.matchesPattern` standard client-side function takes two string expressions as arguments and returns a Boolean value.

The function returns true if the second expression evaluates to an **[ECMAScript]** (JavaScript) regular expression and the result of the first argument expression matches that regular expression, using syntax and semantics of **[ECMAScript]** regular expressions.

Example 75: all non-empty `FirstName` values not containing the letters `b`, `c`, or `d` evaluate to `true`

```
<Apply Function="odata.matchesPattern">
  <Path>FirstName</Path>
  <String>^[^b-d]+$</String>
</Apply>
```

~~14.5.3.1.3~~14.4.4.4 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.~~

Note: string literals are surrounded by single quotes as required by the paren-style key syntax.

Example 76:

```
<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~~

14.4.5 Cast

The ~~edm:Cast~~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]**.

Expression `edm:Cast`

The ~~cast expression~~ `edm:Cast` element MUST ~~specify a `Type`~~contain the `Type` attribute and MUST contain exactly one expression.

~~The cast expression MUST be written with element notation.~~

It MAY contain `edm:Annotation` elements.

Attribute `Type`

The value of `Type` is a qualified type name in scope, or the character sequence `Collection(` followed by the qualified name of a type in scope, followed by a closing parenthesis `)`.

If the specified type is a primitive type or a collection of a primitive type, the facet attributes `MaxLength`, `Unicode`, `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.

Example 77:

```
<Annotation Term="org.example.display.Threshold">
  <Cast Type="Edm.Decimal">
    <Path>Average</Path>
  </Cast>
</Annotation>
```

~~14.5.4.1 Attribute `Type`~~

14.4.6 Collection

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`~~collection expression enables a value to be obtained from zero or more ~~child~~item expressions. The value calculated by the collection expression is the collection of the values calculated by each of the ~~child~~item expressions.

~~The collection expression contains zero or more child expressions.~~ The values of the child expressions MUST all be type compatible.

Expression `edm:Collection`

The ~~collection expression MUST be written with~~`edm:Collection` element ~~notation.~~contains zero or more child expressions.

Example 78:

```
<Annotation Term="org.example.seo.SeoTerms">
  <Collection>
    <String>Product</String>
    <String>Supplier</String>
    <String>Customer</String>
  </Collection>
</Annotation>
```

~~14.5.6 Expression `edm:If`~~

14.4.7 If-Then-Else

The ~~`edm:If`~~if-then-else expression enables a value to be obtained by evaluating a *~~conditional~~condition 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`~~if-then-else expression is a direct child of ~~`edm:Collection` element~~a collection expression, the third child ~~element~~expression MAY be omitted ~~(this~~, reducing it to an if-then expression. This can be used to conditionally add an element to a collection~~)~~.

The first child element is the ~~conditional expression~~condition and MUST evaluate to a ~~Boolean~~Boolean result, e.g. the ~~comparison and logical operators~~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~~expression MUST be evaluated and its value MUST be returned as the result of the ~~`edm:If`~~if-then-else expression. If the ~~conditional~~first 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`~~if-then-else expression. If no third child element is present, nothing is added to the surrounding collection.

Expression `edm:If`

The `edm:If` ~~expression~~element MUST ~~be written with~~contain two or three child expressions that MUST use element notation~~, as shown in the following example~~.

It MAY contain `edm:Annotation` elements.

Example 79:

```
<Annotation Term="org.example.person.Gender">
  <If>
    <Path>IsFemale</Path>
    <String>Female</String>
    <String>Male</String>
  </If>
</Annotation>
```

14.4.8 Is-Of

The is-of expression checks whether the value obtained from its single child expression is compatible with the specified type. It returns `true` if the child expression returns a type that is compatible with the specified type, and `false` otherwise.

~~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`~~element MUST contain the `Type` attribute and MUST 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 64:*~~

```
<Annotation Term="Self.IsPreferredCustomer">
  <IsOf Type="Self.PreferredCustomer">
    <Path>Customer</Path>
  </IsOf>
</Annotation>
```

It MAY contain `edm:Annotation` elements.

~~14.5.7.1~~ Attribute `Type`

~~The `edm:IsOf` expression MUST specify a `Type` attribute whose value is a TypeName in scope.~~

The value of `Type` is the qualified name of a type in scope, or the character sequence `Collection(` followed by the qualified name of a type in scope, followed by a closing parenthesis `)`.

If the specified type is a primitive type or a collection of a primitive type, the facet attributes ~~`MaxLength`~~`MaxLength`, ~~`Precision`~~`Unicode`, `Precision`, ~~`Scale`~~`Scale`, and ~~`SRID`~~`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`~~

Example 80:

```
<Annotation Term="self.IsPreferredCustomer">
  <IsOf Type="self.PreferredCustomer">
    <Path>Customer</Path>
  </IsOf>
</Annotation>
```

14.4.9 Labeled Element

The ~~`edm:LabeledElement`~~labeled element expression assigns a name to ~~a~~its single child expression. The value of the child expression can then be reused elsewhere with ~~an `edm:LabeledElementReference` expression.~~a labeled element reference expression.

A labeled- element expression MUST contain exactly one child expression. The value of the child expression is also the value of the labeled element expression.

A labeled element expression MUST provide a simple identifier value as its name that MUST be unique within the schema containing the expression.

Expression `edm:LabeledElement`

The `edm:LabeledElement` element MUST contain the Name attribute.

It MUST contain a child expression written either in attribute notation or element notation. ~~The value of the child expression is passed through the labeled element expression.~~

~~A~~It MAY contain `edm:Annotation` elements.

Attribute `Name`

The value of `Name` is the labeled~~ element expression MUST be written with element notation~~ element's name.

Example 81:

```
<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.~~

14.4.10 Labeled Element Reference

The labeled element reference expression MUST specify the qualified name of a labeled element expression in scope and returns the value of the identified labeled element expression as its value.

~~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~~qualified name of a labeled element expression in ~~scope~~its body.

~~The labeled element reference expression MUST be written with element notation.~~

~~*Example 66:*~~

Example 82:

```
<Annotation Term="org.example.display.DisplayName">
  <LabeledElementReference>Model.CustomerFirstName</LabeledElementReference>
</Annotation>
```

~~14.5.10 Expression `edm:Null`~~

14.4.11 Null

The ~~`edm:Null`~~null expression returns an untyped null value. The ~~only allowed child elements of the~~ null expression ~~are `edm:Annotation` elements~~MAY be annotated.

The null expression MUST be written with element notation.

~~*Example 67:*~~

~~Expression `edm:Null`~~

The edm:Null element MAY contain `edm:Annotation` elements.

Example 83:


```
<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 types Edm.NavigationPropertyPath, Edm.AnyPropertyPath, or Edm.AnyPath. 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 instance(s) identified by the path.

Example 84:

```
<Annotation Term="@UI.Address">
  <Null>
    <Annotation Term="self.Reason" String="Private" />
  </Null>
</Annotation>
```

14.4.12 Record

The recordThe edm:NavigationPropertyPath expression MAY be provided using element notation or attribute notation.

Example 68:

```
<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 collection-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 collection-valued property.

A path may terminate in a `$count` segment if the previous segment is collection-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.

Paths starting with a forward slash (/) are evaluated starting at the entity container, and the path part after the first forward slash is interpreted relative to the entity container. Paths not starting with a forward slash are interpreted relative to the annotation target, following the rules specified 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 or singleton, 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.

For annotations embedded within or targeting an action, action import, function, or function import, the first segment of a path MUST be a parameter name or `$ReturnType`.

A path expression MAY be provided using element notation or attribute notation.

Example 69:

```
<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 one of the built-in abstract types `Edm.PropertyPath`, `Edm.AnyPropertyPath`, or `Edm.AnyPath`. 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 70:

```
<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 MAY specify the structured type if its result, which MUST resolve to an entity type or complex type in scope. If not explicitly specified, the type is derived from the expression's context.

A record expression contains zero or more `edm:PropertyValue` elements.property value expressions. For each single-valued structural or navigation property of the record construct'sexpression's type that is neither nullable nor specifies a default value an `edm:PropertyValue` child elementa property value expression MUST be provided. The only exception is if the record expression is the direct childvalue of an `edm:Annotation` elementannotation for a term that has a base termbase 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~~a property value expression is equivalent to specifying ~~a child element with~~ an empty collection as its value.

~~A record~~Expression `edm:Record`

The `edm:Record` element MAY contain the `Type` attribute and MAY contain `edm:PropertyValue` elements.

It MAY contain `edm:Annotation` elements.

Attribute `Type`

The value of `Type` is the qualified name of a structured type in scope.

Element `edm:PropertyValue`

The `edm:PropertyValue` element MUST contain the `Property` attribute, and it MUST contain exactly one expression ~~MUST~~that MAY be ~~written with~~provided using either element notation~~, as shown in the following example~~ or attribute notation.

It MAY contain `edm:Annotation` elements.

Attribute `Property`

The value of `Property` is the name of a property of the type of the enclosing `edm:Record` expression.

Example 85: record with two structural and two navigation properties

```
<Annotation Term="org.example.person.Employee">
  <Record>
    <Annotation Term="Core.Description" String="Annotation on record" />
    <PropertyValue Property="GivenName" Path="FirstName" />">
      <Annotation Term="Core.Description"
                  String="Annotation on record member" />
    </PropertyValue>
    <PropertyValue Property="Surname" Path="LastName" />
    <PropertyValue Property="Manager" Path="DirectSupervisor" />
    <PropertyValue Property="CostCenter">
      <UrlRef>
        <Apply Function="odata.fillUriTemplate">
          <String>http://host/anotherservice/CostCenters('{ccid}')</String>
          <LabeledElement Name="ccid" Path="CostCenterID" />
        </Apply>
      </UrlRef>
    </PropertyValue>
  </Record>
</Annotation>
```

~~14.5.14.1 Attribute `Type`~~

~~A record expression MAY specify a QualifiedName value for the `Type` attribute that MUST resolve to an entity type or complex type in scope. If no value is specified for the type attribute, the type is derived from the expression's context.~~

~~14.5.14.2 Element `edm:PropertyValue`~~

14.4.13 URL Reference

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`~~URL reference expression enables a value to be obtained by sending a `GET` request ~~to the value of the `UrlRef` expression~~.

The ~~`edm:UrlRef` element~~URL reference expression MUST contain exactly one expression of type `Edm.String`. ~~The `edm:UrlRef` expression MAY be provided using element notation or attribute notation.~~

~~The~~Its value is treated as a URL ~~may~~that MAY be relative or absolute; relative URIs are relative to the ~~`xml:`~~URL of the document containing the URL reference expression, or relative to a base ~~attribute, see **[XML-Base]**.~~URL specified in a format-specific way.

The response body of the `GET` request MUST be returned as the result of the ~~`edm:UrlRef`~~URL reference expression. The result of the `edm:UrlRef` expression MUST be type compatible with the type expected by the surrounding element or expression.

~~*Example 72:*~~

> ### Expression `edm:UrlRef`
>
> The `edm:UrlRef` expression MAY be provided using element notation or attribute notation.
>
> Relative URLs are relative to the `xml:base` attribute, see **[XML-Base]**.
>
> In element notation it MAY contain `edm:Annotation` elements.

Example 86:

```
<Annotation Term="~~Vocab~~org.example.person.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~~ CSDL Examples

15 Identifier and Path Values

15.1 Namespace

A namespace is a dot-separated sequence of simple identifiers with a maximum length of 511 Unicode characters.

15.2 Simple Identifier

A simple identifier is a Unicode character sequence with the following restrictions:

- It consists of at least one and at most 128 Unicode characters.
- The first character MUST be the underscore character (U+005F) or any character in the Unicode category “Letter (L)” or “Letter number (Nl)”.
- The remaining characters MUST be the underscore character (U+005F) or any character in the Unicode category “Letter (L)”, “Letter number (Nl)”, “Decimal number (Nd)”, “Non-spacing mark (Mn)”, “Combining spacing mark (Mc)”, “Connector punctuation (Pc)”, and “Other, format (Cf)”.

Non-normatively speaking it starts with a letter or underscore, followed by at most 127 letters, underscores or digits.

15.3 Qualified Name

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.

15.4 Target Path

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 qualified name 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 87: Target expressions

```
MySchema.MyEntityContainer/MyEntitySet
MySchema.MyEntityContainer/MySingleton
MySchema.MyEntityContainer/MyEntitySet/MyContainmentNavigationProperty
MySchema.MyEntityContainer/MyEntitySet/My.EntityType/MyContainmentNavProperty
MySchema.MyEntityContainer/MySingleton/MyComplexProperty/MyContainmentNavProp
```


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.

~~15.1~~16.1 ~~Products and Categories Example~~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~~" />~~">
      <Annotation Term="Core.DefaultNamespace" />
    </edmx:Include>
  </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~~Measures" 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~~Measures.ISOCurrency" Path="Currency" />
        </Property>
        <Property Name="Currency" Type="Edm.String" MaxLength="3" />
        <NavigationProperty Name="Category" Type="ODataDemo.Category"
                            Nullable="false" Partner="Products" />
        <NavigationProperty Name="Supplier" Type="ODataDemo.Supplier"
                            Partner="Products" />
      </EntityType>
      <EntityType Name="Category">
        <Key>
         <PropertyRef Name="ID" />
        </Key>
        <Property Name="ID" Type="Edm.Int32" Nullable="false" />
        <Property Name="Name" Type="Edm.String">
          <Annotation Term="Core.IsLanguageDependent" />
        </Property>
        <NavigationProperty Name="Products" Partner="Category"
                            Type="Collection(ODataDemo.Product)">
          <OnDelete Action="Cascade" />
```



```
      </NavigationProperty>
    </EntityType>
    <EntityType Name="Supplier">
      <Key>
        <PropertyRef Name="ID" />
      </Key>
      <Property Name="ID" Type="Edm.String" Nullable="false" />
      <Property Name="Name" Type="Edm.String" />
      <Property Name="Address" Type="ODataDemo.Address" Nullable="false" />
      <Property Name="Concurrency" Type="Edm.Int32" Nullable="false" />
      <NavigationProperty Name="Products" Partner="Supplier"
                          Type="Collection(ODataDemo.Product)" />
    </EntityType>
    <EntityType Name="Country">
      <Key>
        <PropertyRef Name="Code" />
      </Key>
      <Property Name="Code" Type="Edm.String" MaxLength="2"
                            Nullable="false" />
      <Property Name="Name" Type="Edm.String" />
    </EntityType>
    <ComplexType Name="Address">
      <Property Name="Street" Type="Edm.String" />
      <Property Name="City" Type="Edm.String" />
      <Property Name="State" Type="Edm.String" />
      <Property Name="ZipCode" Type="Edm.String" />
      <Property Name="CountryName" Type="Edm.String" />
      <NavigationProperty Name="Country" Type="ODataDemo.Country">
        <ReferentialConstraint Property="CountryName"
                               ReferencedProperty="Name" />
      </NavigationProperty>
    </ComplexType>
    <Function Name="ProductsByRating">
      <Parameter Name="Rating" Type="Edm.Int32" />
      <ReturnType Type="Collection(ODataDemo.Product)" />
    </Function>
    <EntityContainer Name="DemoService">
      <EntitySet Name="Products" EntityType="ODataDemo.Product">
        <NavigationPropertyBinding Path="Category" Target="Categories" />
      </EntitySet>
      <EntitySet Name="Categories" EntityType="ODataDemo.Category">
        <NavigationPropertyBinding Path="Products" Target="Products" />
      </EntitySet>
      <EntitySet Name="Suppliers" EntityType="ODataDemo.Supplier">
        <NavigationPropertyBinding Path="Products" Target="Products" />
        <NavigationPropertyBinding Path="Address/Country"
                                   Target="Countries" />
        <Annotation Term="Core.OptimisticConcurrency">
          <Collection>
            <PropertyPath>Concurrency</PropertyPath>
          </Collection>
        </Annotation>
      </EntitySet>
      <Singleton Name="MainSupplier" 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>
```


~~15.2~~16.2 ~~Annotations for Products and Categories Example~~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>
```


16 Attribute Values

16.1 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.

16.2 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>
```


17 Conformance

Non-normatively speaking it starts with a letter or underscore, followed by at most 127 letters, underscores or digits.

16.3 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.

16.4 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.

16.5 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 75: Target expressions

```
MySchema.MyEntityContainer/MyEntitySet
MySchema.MyEntityContainer/MySingleton
MySchema.MyEntityContainer/MyEntitySet/MyContainmentNavigationProperty
MySchema.MyEntityContainer/MyEntitySet/My.EntityType/MyContainmentNavProperty
MySchema.MyEntityContainer/MySingleton/MyComplexProperty/MyContainmentNavProp
```

16.6 Boolean

One of the literals `true` or `false`.

~~17~~ Conformance

Conforming services MUST follow all rules of this specification document for the types, sets, functions, actions, containers and annotations they expose.

In addition, conforming services MUST NOT return 4.01 ~~elements or attributes, or new values for elements or attributes with enumerated values,~~CSDL constructs for requests made with `OData-MaxVersion:4.0`.

Specifically, they

1. MUST NOT include properties in derived types that overwrite a property defined in the base type
2. MUST NOT include `Edm.Untyped`
3. ~~MUST NOT include extended `Edm.Path` expression~~
3. MUST NOT use path syntax added with 4.01
4. MUST NOT use ~~`Edm.AnyPath`~~`Edm.ModelElementPath` and ~~`Edm.AnyPropertyPath`~~`Edm.AnyPropertyPath`
5. MUST NOT specify ~~referential constraints~~referential constraints to complex types and navigation properties
6. MUST NOT include a non-abstract entity type with no inherited or defined ~~entity key~~entity key
7. MUST NOT include the `Core.DefaultNamespace` annotation on ~~the `edm:Include` element~~included schemas
8. MUST NOT return the Unicode facet for terms, parameters, and return types
9. MUST NOT include ~~Collections~~collections of ~~`Edm.ComplexType`~~`Edm.ComplexType` or ~~`Edm.Untyped`~~`Edm.Untyped`
10. MUST NOT specify a key as a property of a related entity
11. SHOULD NOT include new/unknown values for the `AppliesTo` attribute
12. SHOULD specify the Nullable facet for collections
~~12.~~13. MAY include new CSDL annotations

In addition, to comply with OData 4.01, services:

~~13.~~14. MUST specify the ~~Nullable~~Nullable facet for collections

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~~

Appendix A. Acknowledgments

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

Appendix B. ~~Revision History~~Table of XML Elements and Attributes

Appendix C. References

~~Appendix B.~~

Revision	Date	Editor	Changes Made
Working Draft 01	2016-09-07	Michael Pizzo Ralf Handl	Imported content from 4.0 Errata 3 specification and integrated initial 4.01 features
Committee Specification Draft 01	2016-12-08	Michael Pizzo Ralf Handl	Integrated 4.01 features
Committee Specification Draft 02	2017-06-08	Michael Pizzo Ralf Handl	Incorporated normative text from former OData Part 3: CSDL