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

Committee Specification Draft 01

26 April 2013

Specification URIs

This version:

http://docs.oasis-open.org/odata/odata/v4.0/csd01/part3-csdl/odata-v4.0-csd01-part3-csdl.doc (Authoritative)
http://docs.oasis-open.org/odata/odata/v4.0/csd01/part3-csdl/odata-v4.0-csd01-part3-csdl.html
http://docs.oasis-open.org/odata/odata/v4.0/csd01/part3-csdl/odata-v4.0-csd01-part3-csdl.pdf

Previous version:

N/A

Latest version:

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

Technical Committee:

OASIS Open Data Protocol (OData) TC

Chairs:

Barbara Hartel (barbara.hartel@sap.com), SAP AG
Ram Jeyaraman (Ram.Jeyaraman@microsoft.com), Microsoft

Editors:

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

Additional artifacts:

This prose specification is one component of a Work Product which consists of:

- *OData Version 4.0 Part 1: Protocol*. http://docs.oasis-open.org/odata/odata/v4.0/csd01/part1-protocol/odata-v4.0-csd01-part1-protocol.html
- *OData Version 4.0 Part 2: URL Conventions*. http://docs.oasis-open.org/odata/odata/v4.0/csd01/part2-url-conventions/odata-v4.0-csd01-part2-url-conventions.html
- *OData Version 4.0 Part 3: Common Schema Definition Language (CSDL)* (this document). http://docs.oasis-open.org/odata/odata/v4.0/csd01/part3-csdl/odata-v4.0-csd01-part3-csdl.html
- ABNF components: *OData ABNF Construction Rules Version 4.0* and *OData ABNF Test Cases.* http://docs.oasis-open.org/odata/odata/v4.0/csd01/abnf/
- Vocabulary components: *OData Core Vocabulary* and *OData Measures Vocabulary*. http://docs.oasis-open.org/odata/odata/v4.0/csd01/vocabularies/
- XML schemas: OData EDMX XML Schema and OData EDM XML Schema. http://docs.oasis-open.org/odata/odata/v4.0/csd01/schemas/

- OData Metadata Service Entity Model: http://docs.oasis-open.org/odata/odata/v4.0/csd01/models/MetadataService.edmx

Related work:

This specification is related to:

- *OData Atom Format Version 4.0*. Latest version. http://docs.oasis-open.org/odata/odata-atom-format/v4.0/odata-atom-format-v4.0.html.
- *OData JSON Format Version 4.0*. Latest version. http://docs.oasis-open.org/odata/odata-json-format/v4.0/odata-json-format-v4.0.html.

Declared XML namespaces:

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

Abstract:

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

Status:

This document was last revised or approved by the OASIS Open Data Protocol (OData) TC on the above date. The level of approval is also listed above. Check the "Latest version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committees/odata/.

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

Citation format:

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

[OData-Part3]

OData Version 4.0 Part 3: Common Schema Definition Language (CSDL). 26 April 2013. OASIS Committee Specification Draft 01. http://docs.oasis-open.org/odata/odata/v4.0/csd01/part3-csdl/odata-v4.0-csd01-part3-csdl.html.

Notices

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

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

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

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

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

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

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

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

Table of Contents

1 Introduction

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

1.1 Terminology

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

1.2 Normative References

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

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

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

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

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

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

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

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

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

[RFC6570] Gregorio, J. et al., “URI Template”, RFC 6570, March 2012. http://tools.ietf.org/html/rfc6570.

[XML-1.1] Extensible Markup Language (XML) 1.1 (Second Edition) , Paoli, J., Sperberg-McQueen, C. M., Yergeau, F., Cowan, J., Maler, E., Bray, T., 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) , Marsh, J., Tobin, R., 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 , Sperberg-McQueen, C. M., Beech, D., Gao, S., Maloney, M., Mendelsohn, N., Thompson, H. S., 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 , Malhotra, A., Sperberg-McQueen, C. M., Gao, S., Thompson, H. S., Peterson, D., Biron, P. V., 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/.

2 CSDL Namespaces

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

2.1 Namespace EDMX

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

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

Prior versions of OData used the following namespace for EDMX:

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

They are non-normative for this specification.

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

2.2 Namespace EDM

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

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

Prior versions of CSDL used the following namespaces for EDM:

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

They are non-normative for this specification.

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

2.3 XML Schema Definitions

This specification contains normative XML schemas for the EDMX and EDM namespaces; see links in the “Additional artifacts” section on the cover page.

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

3 Entity Model Wrapper

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

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

The *service* consists of all entity containers defined in the metadata document. These entity containers MAY extend entity containers defined in referenced documents.

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

3.1 Element `edmx:Edmx`

The metadata document MUST contain a single 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. Reference elements specify the location of schemas used by the OData service.

The following example demonstrates the basic structure of the `edmx:Edmx` element and the `edmx:DataServices` element:

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

3.1.1 Attribute `Version`

The `Version` attribute MUST be present on the `edmx:Edmx` element.

The `Version` attribute is a string value that specifies the version of the EDMX wrapper, and must be of the form `<majorversion>.<minorversion>`. This version of the specification defines version `4.0` of the EDMX Wrapper.

3.2 Element `edmx:DataServices`

The `edmx:DataServices` element contains zero or more `edm:Schema` elements which define the schema(s) 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 *scope* of a CSDL document is the document itself and all schemas included from directly referenced documents. All entity types, complex types and other named elements *in scope* (that is, defined in the document itself or a schema of a directly referenced document) can be accessed from a referencing document by their namespace-qualified names.

Referencing another document may alter the model defined by the referencing document. For instance, if a referenced document defines an entity type derived from an entity type in the referencing document, then an entity set of the service defined by the referencing document may return entities of the derived type. This is identical to the behavior if the derived type had been defined directly in the referencing document.

Note: referencing documents is not recursive. Only named elements defined in directly referenced documents can be accessed; elements that are defined in documents that are only referenced by referenced documents cannot be accessed.

The following example demonstrates usage of the reference element to reference entity models that contain 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:Edmx>
```

3.3.1 Attribute `Uri`

The `edmx:Reference` element MUST specify a `Uri` attribute. The `Uri` attribute uniquely identifies a model. The value of the `Uri` attribute SHOULD be 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.

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.

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.

The following example demonstrates using the edmx:IncludeAnnotations element to reference documents that contain annotations:

The following example demonstrates using the edmx:IncludeAnnotations element to 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:Reference>
 <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.

3.5.1 Attribute TermNamespace

An edmx:IncludeAnnotations element MUST provide a Namespace value for the TermNamespace attribute. A term namespace is a string that disambiguates terms with the same name.

For instance, assume both org.schema and org.microformats define a term named Address. Although the terms have the same name, they are uniquely identifiable since each term is in a model with a unique namespace.

If a value is supplied, the include element will import the set of annotations that apply terms from the namespace in the value. The term namespace attribute also provides consumers insight about what namespaces are used in the annotations document. If there are no include elements that have a term namespace of interest to the consumer, the consumer can opt to not 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 in the specified TermNamespace with the specified Qualifier SHOULD be applied. If Qualifier is not specified, all annotations within the document from the specified namespace SHOULD be applied.

4 Common Characteristics of Entity Models

4.1 Nominal Types

A nominal type has a name. The name MUST be a SimpleIdentifier. Prefixed with a Namespace and a dot (.) this produces a fully qualified name of the form QualifiedName. The qualified type name MUST be unique within a model as it facilitates references to the element from other parts of the model.

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

- Namespace-qualified name
- Alias-qualified name

Consider the following example:

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

The various ways of referring to the nominal type are:

- References in any namespace can use the fully qualified name, for example, `org.example.Address`
- References in any namespace can specify an alias and use an alias-qualified name, for example, `sales.Address`

4.2 Structured Types

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

4.3 Structural Properties

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

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

4.4 Primitive Types

Structured types are composed of other structured types and primitive types. CSDL defines the following fully qualified primitive types:

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

Type	Meaning
Edm.DateTimeOffset	Date and time with a time-zone offset, no leap seconds
Edm.Decimal	Numeric values with fixed precision and scale
Edm.Double	Floating-point number with 15 digits precision
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	Floating-point number with 7 digits precision
Edm.Stream	Fixed-length or variable-length data stream
Edm.String	Fixed-length or variable-length sequence of UTF-8 characters
Edm.TimeOfDay	Clock time 0-23:59:59.999999999999
Edm.Geography	Abstract base type for all Geography types
Edm.GeographyPoint	A point in a round-earth coordinate system
Edm.GeographyLineString	Line string in a round-earth coordinate system
Edm.GeographyPolygon	Polygon in a round-earth coordinate system
Edm.GeographyMultiPoint	Collection of points in a round-earth coordinate system
Edm.GeographyMultiLineString	Collection of line strings in a round-earth coordinate system
Edm.GeographyMultiPolygon	Collection of polygons in a round-earth coordinate system
Edm.GeographyCollection	Collection of arbitrary Geography values
Edm.Geometry	Abstract base type for all Geometry types
Edm.GeometryPoint	Point in a flat-earth coordinate system
Edm.GeometryLineString	Line string in a flat-earth coordinate system
Edm.GeometryPolygon	Polygon in a flat -earth coordinate system
Edm.GeometryMultiPoint	Collection of points in a flat -earth coordinate system
Edm.GeometryMultiLineString	Collection of line strings in a flat -earth coordinate system
Edm.GeometryMultiPolygon	Collection of polygons in a flat -earth coordinate system
Edm.GeometryCollection	Collection of arbitrary Geometry values

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

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

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

4.5 Built-In Abstract Types

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

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

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

- `Edm.EntityType`
  - cannot be used as the type of a named entity in an entity container because it doesn't define a structure, which defeats the purpose of a named entity.
  - 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.
- `Collection(Edm.PrimitiveType)` and `Collection(Edm.ComplexType)`
  - cannot be used as the type of a property.
  - Cannot be used as the return type of a function.

Vocabulary terms can, in addition, use

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

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

4.6 Annotations

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

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

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

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

5 Schema

One or more schemas describe the entity model exposed by an OData service. The schema acts as a container for all of the entity types, complex types and other parts of the entity model.

5.1 Element edm:Schema

The edm:Schema is the root of an entity model exposed by an OData service. Although an edmx:DataServices element can contain more than one Schema elements, most OData services will contain exactly one schema.

An edm:Schema element contains zero or more of the following elements:

- edm:Action
- edm:Annotations
- 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 action overloads and function overloads. The names are local to the schema; they need not be unique within a document.

5.1.1 Attribute Namespace

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

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

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

5.1.2 Attribute Alias

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

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

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

6 Property

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

For example, the following complex type has 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 allow properties to be added dynamically. When requesting the value of a missing property from an open entity type, the instance MUST return a representation of the `null` value.

6.1 Element `edm:Property`

An `edm:Property` element allows the construction of structured types from structural properties.

For instance, the following property could be used to hold zero or more strings representing the names of measurement units:

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

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

6.1.1 Attribute `Name`

A property MUST specify a SimpleIdentifier value for the `Name` attribute. The name attribute allows a name to be assigned to the property. This name is used when serializing or deserializing OData payloads and can be used for other purposes, such as code generation.

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

6.1.2 Attribute `Type`

A property MUST specify a value for the `Type` attribute. The value of this attribute determines the type for the value of the property on instances of the containing type.

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

6.2 Property Facets

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

Facets apply to the type referenced in the element where the facet is declared. If the type is a collection type declared with attribute notation, the facets apply to the types in the collection. In the following example, the `Precision` facet applies to the `DateTimeOffset` type.

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

6.2.1 Attribute Nullable

Any property MAY define a Boolean value for the Nullable facet attribute. The value of this attribute determines whether a value is required for the property on instances of the containing type.

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

6.2.2 Attribute MaxLength

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

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

6.2.3 Attribute Precision

A datetimeoffset, decimal, duration, or timeofday property MAY define a value for the Precision attribute.

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

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

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

6.2.4 Attribute Scale

A decimal property MAY define a non-negative integer value or variable for the Scale attribute. The integer value of this attribute specifies the maximum number of digits allowed to the right of the decimal point. The value variable means that the number of digits to the right of the decimal point may vary from zero to the value of the Precision attribute.

The value of the Scale attribute MUST be less than or equal to the value of the Precision attribute.

If no value is specified, the Scale facet defaults to zero.

6.2.5 Attribute Unicode

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

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

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

6.2.6 Attribute SRID

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

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

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

6.2.7 Attribute `DefaultValue`

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

Default values MUST be represented according to the `xxxBody` rule defined in **[OData-ABNF]** that is appropriate for the type of the property.

If no value is specified, the `DefaultValue` attribute defaults to `null`.

7 Navigation Property

7.1 Element edm:NavigationProperty

A navigation property allows navigation to related entities.

In the following example, the Product entity type has a navigation property to a Category, which has a navigation link back to one or more products:

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

7.1.1 Attribute Name

The navigation property MUST provide a SimpleIdentifier value for the `Name` attribute. The name attribute is a meaningful string that characterizes the relationship when navigating from the structured type that declares the navigation property to the related entity type.

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

7.1.2 Attribute Type

A navigation property MUST specify a value for the `Type` attribute. The value of the type attribute MUST resolve to an entity type or a collection of an entity type in scope, i.e. either declared in the same document or a document referenced with an `edm:Reference` element.

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

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

7.1.3 Attribute Nullable

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

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

7.1.4 Attribute Partner

A navigation property of an entity type MAY specify a SimpleIdentifier value for the `Partner` attribute. If specified, the value of this attribute MUST be the name of a direct or inherited navigation property in the

entity type specified in the `Type` attribute. The type of the partner navigation property MUST be the containing entity type of the current navigation property or one of its parent entity types.

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

7.1.5 Attribute `ContainsTarget`

A navigation property of an entity type MAY assign a Boolean value to the `ContainsTarget` attribute. If no value is assigned to the `ContainsTarget` attribute, the attribute defaults to `false`. If the value assigned to the `ContainsTarget` attribute is `true`, entities of the entity type that specifies the navigation property contain the entities referenced by the navigation property.

It MUST NOT be possible for an entity type to contain itself by following more than one containment navigation property.

When a navigation property with `ContainsTarget="true"` navigates between entity types in the same entity set it is called recursive containment. If the containment is recursive, the partner navigation property MUST be nullable and specify a single entity type (i.e. have a multiplicity of `0..1`). If the containment is not recursive, the partner navigation property MUST NOT be nullable (i.e. have a multiplicity of `1`).

If the containment is recursive, a navigation property binding for the containment navigation property MUST specify the same entity set that encloses the navigation property binding.

An entity cannot be contained by more than one entity, so an entity set MUST NOT be specified in the `EntitySet` attribute of a navigation property binding for more than one containment navigation property.

7.2 Element `edm:ReferentialConstraint`

A navigation property whose `Type` attribute specifies a single entity type MAY define a referential constraint. A referential constraint asserts that if the navigation property is not null, the property of the *dependent* entity (the source of the navigation) listed in the referential constraint MUST have the same values as the referenced property of the *principal* entity (the target of the navigation).

In the example that follows, the category must exist for a product in that category to exist, and the category ID of the product is identical to the ID of the category:

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

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

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

7.3 Element `edm:OnDelete`

A navigation property MAY define an `edm:OnDelete` element. It prescribes the action that should be taken when the (last) entity targeted by the navigation property is deleted.

If present, the `edm:OnDelete` element MUST define a value for the `Action` attribute. The value assigned to the action attribute MUST be

- `Cascade`, meaning the dependent entities will be deleted if the principal entity is deleted,

- `None`, meaning a `DELETE` operation on a principal entity with dependent entities will fail,
- `SetNull`, meaning all dependent properties that do not participate in other referential constraints will be set to null,
- `SetDefault`, meaning all dependent properties that do not participate in other referential constraints will be set to their default value.

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

8 Entity Type

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

A key MUST be supplied if and only if the entity type does not specify a base type. The key consists of one or more references to structural properties of the entity type.

An entity type can define two types of properties. A structural property is a named reference to a primitive, complex, or enumeration type, or a collection of primitive, complex, or enumeration types. A navigation property is a named reference to another entity type or collection of entity types. All properties MUST have a unique name within an entity type. Properties MUST NOT have the same name as the declaring entity type.

An open entity type allows properties to be added to an instance of the type dynamically. Any request for the value of a missing property on an open entity type MUST return `null`.

A simple example of an entity type is as follows:

```
<EntityType Name="Product">
 <Key>
  <PropertyRef Name="ID"/>
 </Key>
 <Property Name="ID" Type="Edm.Int32" Nullable="false"/>
 <Property Name="Name" Type="Edm.String" Nullable="true" />
 <Property Name="Description" Type="Edm.String" Nullable="true" />
 <NavigationProperty Name="Category" Type="Self.Category"/>
 <NavigationProperty Name="Supplier" Type="Self.Supplier"/>
</EntityType>
```

The following example shows an entity type based on the previous example:

```
<EntityType Name="DiscontinuedProduct" BaseType="Self.Product">
 <Property Name="DiscontinuedDate" Type="Edm.DateTimeOffset" Nullable="true"/>
</EntityType>
```

8.1 Element `edm:EntityType`

The `edm:EntityType` element represents an entity type in the entity model. It MAY contain zero or more `edm:Property` elements and zero or more `edm:NavigationProperty` elements.

It also MAY contain one `edm:Key` element.

8.1.1 Attribute `Name`

An entity type MUST provide a SimpleIdentifier value for the `Name` attribute because it is a nominal type. The value identifies the entity type and MUST be unique within its namespace.

8.1.2 Attribute `BaseType`

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

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

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

8.1.3 Attribute Abstract

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

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

8.1.4 Attribute OpenType

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

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

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

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

8.1.5 Attribute HasStream

An entity type MAY specify a Boolean value for the `HasStream` attribute.

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

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

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

8.2 Element edm:Key

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

An entity type's key refers to the set of properties that uniquely identify an instance of the entity type within an entity set. If specified, the key MUST contain one or more `edm:PropertyRef` elements. An `edm:PropertyRef` element references an `edm:Property`. The properties that compose the key MUST be non-nullable and typed with an enumeration type, one of the following primitive types, or a type definition based on one of these primitive types:

- `Edm.Boolean`
- `Edm.Byte`
- `Edm.Date`
- `Edm.DateTimeOffset`
- `Edm.Decimal`
- `Edm.Duration`
- `Edm.GeographyPoint`
- `Edm.GeometryPoint`
- `Edm.Guid`
- `Edm.Int16`
- `Edm.Int32`
- `Edm.Int64`
- `Edm.SByte`
- `Edm.String`

- `Edm.TimeOfDay`

The following entity type has 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" Nullable="true" />
</EntityType>
```

The following entity type has a simple key referencing a property of a complex type:

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

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

The following entity type has a composite key:

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

8.3 Element `edm:PropertyRef`

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

8.3.1 Attribute `Name`

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

8.3.2 Attribute `Alias`

The `edm:PropertyRef` element MAY define a SimpleIdentifier value for the `Alias` attribute.

The `Alias` attribute defines an alias for the property identified by the `Name` attribute. The alias 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 be defined if the key property is a member of a complex type.

If an alias is defined, it MUST be used in the key predicate of URLs instead of the value assigned to the `Name` attribute.

Based on the example above requests to an entity set Categories of type Category can be send:

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

The alias MUST NOT be used in the query part, e.g. in filter expressions.

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


9 Complex Type

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

All properties MUST have a unique name. Properties MUST NOT have the same name as the declaring complex type.

The following example demonstrates a complex type that is used by two entity types:

```
<ComplexType Name="Dimensions">
 <Property Name="Height" Nullable="false" Type="Edm.Decimal"/>
 <Property Name="Weight" Nullable="false" Type="Edm.Decimal"/>
 <Property Name="Length" Nullable="false" Type="Edm.Decimal"/>
</ComplexType>
<EntityType Name="Product">
 ...
 <Property Name="ProductDimensions" Type="Self.Dimensions"/>
 <Property Name="ShippingDimensions" Type="Self.Dimensions"/>
</EntityType>
<EntityType Name="ShipmentBox">
 ...
 <Property Name="Dimensions" Type="Self.Dimensions"/>
</EntityType>
```

9.1 Element edm:ComplexType

The `edm:ComplexType` element represents a complex type in an entity model.

If no base type is specified, the edm:ComplexType element MUST contain one or more `edm:Property` elements describing the properties of the complex type.

If a base type is specified, the edm:ComplexType element MAY contain zero or more `edm:Property` elements describing *additional* properties of the derived complex type.

9.1.1 Attribute Name

A complex type MUST provide a SimpleIdentifier value for the `Name` attribute because a it is a nominal type. The value identifies the complex type and MUST be unique within its namespace.

9.1.2 Attribute BaseType

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

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

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

9.1.3 Attribute Abstract

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

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

9.1.4 Attribute `OpenType`

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

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

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

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

10 Enumeration Type

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

Enumeration types typically allow the selection of a single member. The `IsFlags` attribute allows entity model authors to indicate that more than one value can be selected.

The following example shows a simple flags-enabled enum:

```
<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.

10.1.1 Attribute `Name`

An enumeration type MUST provide a SimpleIdentifier value for the `Name` attribute because it is a nominal type. The value identifies the enumeration type and MUST be unique within its namespace.

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

10.1.2 Attribute `UnderlyingType`

An enumeration type has an underlying type which specifies the allowable values for member mapping.

An enumeration type MAY include an `UnderlyingType` attribute to specify an underlying type whose value MUST be one of `Edm.Byte`, `Edm.SByte`, `Edm.Int16`, `Edm.Int32`, or `Edm.Int64`. If the `UnderlyingType` attribute is not specified, a 32-bit integer MUST be used as the underlying type.

10.1.3 Attribute `IsFlags`

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

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

10.2 Element `edm:Member`

An enumeration type typically has two or more members. Members represent discrete options for the enumeration type.

Enumeration members are declared with the `edm:Member` element.

For example, the following enumeration type has three discrete members:

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

10.2.1 Attribute Name

Each enumeration member MUST provide a SimpleIdentifier value for the Name attribute. The enumeration type MUST NOT declare two members with the same name.

10.2.2 Attribute Value

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

If the IsFlags attribute has a value of false and the Value attribute is not explicitly set, the value MUST be assigned to 0 for the first member or one plus the previous member value for any subsequent members.

If the IsFlags attribute has a value of true, the Value attribute MUST be explicitly set, and it MUST be assigned a non-negative number. 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.

In the example that follows, FirstClass MUST be assigned a value of 0, TwoDay a value of 4, and Overnight a value of 5.

```
<EnumType Name="ShippingMethod">
 <Member Name="FirstClass"/>
 <Member Name="TwoDay" Value="4"/>
 <Member Name="Overnight"/>
</EnumType>
```

In the next example pattern values can be combined, and some combined values have explicit names:

```
<EnumType Name="Pattern" UnderlyingType="Edm.Int32" IsFlags="true">
   <Member Name="Plain" Value="0"/>
   <Member Name="Red" Value="1"/>
   <Member Name="Blue" Value="2"/>
   <Member Name="Yellow" Value="4"/>
   <Member Name="Solid" Value="8"/>
   <Member Name="Striped" Value="16"/>
   <Member Name="SolidRed" Value="9"/>
   <Member Name="SolidBlue" Value="10"/>
   <Member Name="SolidYellow" Value="12"/>
   <Member Name="RedBlueStriped" Value="19"/>
   <Member Name="RedYellowStriped" Value="21"/>
   <Member Name="BlueYellowStriped" Value="22"/>
</EnumType>
```

11 Type Definition

11.1 Element edm:TypeDefinition

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

Type definitions can be used wherever a primitive type is used (other than as the underlying type in a new type definition), and are type-comparable with their underlying types and any type definitions defined using the same underlying type.

11.1.1 Attribute Name

A type definition MUST provide a SimpleIdentifier value for the `Name` attribute because it is a nominal type. The value identifies the type definition and MUST be unique within its namespace.

11.1.2 Attribute UnderlyingType

The `edm:TypeDefinition` element MUST provide the QualifiedName of a primitive type as the value of the `UnderlyingType` attribute. It MUST NOT provide the name of a type definition as the value of the `UnderlyingType` attribute.

11.1.3 Type Definition Facets

The `edm:TypeDefinition` element MAY specify values for zero or more 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. Applying the same annotation to a property whose type definition already defines that annotation is an error.

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:

```
<TypeDefinition Name="Length" UnderlyingType="Edm.Int32">
  <Annotation Term="Org.Odata.Measurements.V1.Unit"
              String="Centimeters"/>
</TypeDefinition>
<TypeDefinition Name="Weight" UnderlyingType="Edm.Int32">
  <Annotation Term="Org.Odata.Measurements.V1.Unit"
              String="Kilograms"/>
</TypeDefinition>
<ComplexType Name="Size">
  <Property Name="Height" Type="Self.Length"/>
  <Property Name="Weight" Type="Self.Weight"/>
</ComplexType>
```

12 Action and Function

12.1 Element edm:Action

The edm:Action element is a nominal type that represents an Action in an entity model.

Actions MAY have observable side-effects and MAY return a single instance or a collection of instances of any type. Actions are not composable.

The action MAY specify a return type using the edm:ReturnType element. The return type must be a scalar, entity or complex type, or a collection of scalar, entity or complex types.

The action may also define zero or more edm:Parameter elements to be used during the execution of the action.

A bindable action MAY have overloads, that is multiple edm:Action elements in a schema MAY specify the same value for the Name attribute. The combination of the action name and the binding parameter type MUST identify the action overload.

12.1.1 Attribute Name

An action MUST provide a SimpleIdentifier value for the Name attribute. For non-bindable actions the name MUST be unique within its namespace. For bindable actions the combination of the action name and the binding parameter type MUST be unique within its namespace.

12.1.2 Attribute ReturnType

If the return type is written with attribute notation, a TypeName value MUST be provided for the ReturnType attribute.

If a value is provided for the ReturnType attribute, the edm:Action element MUST NOT contain an edm:ReturnType element.

12.1.3 Attribute IsBindable

An action element MAY specify a Boolean value for the IsBindable attribute. If no value is specified for the IsBindable attribute, the value defaults to false.

If the value of the IsBindable attribute is set to true, the action element MUST contain at least one edm:Parameter element, and the first parameter is the binding parameter. It MAY be of any type, and it MAY be nullable.

12.1.4 Attribute EntitySetPath

Bindable actions MAY specify a value for the EntitySetPath attribute if determination of the entity set for the return type is contingent on the binding parameter.

The value for the EntitySetPath attribute consists of a series of segments joined together with forward slashes.

The first segment of the entity set path MUST be the name of the binding parameter. The remaining segments of the entity set path MUST represent navigation segments or type casts.

A navigation segment names the SimpleIdentifier of the navigation property to be traversed. A type cast segment names the QualifiedName of the entity type that should be returned from the type cast.

12.2 Element edm:Function

The edm:Function element is a nominal type that 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 MAY specify a return type using the edm:ReturnType element. The return type must be a scalar, entity or complex type, or a collection of scalar, entity or complex types.

The function may also define zero or more edm:Parameter elements to be used during the execution of the function.

A function MAY have overloads, that is multiple edm:Function elements in a schema MAY specify the same value for the Name attribute. The combination of the function name and the unordered list of parameter names and types MUST identify a particular function overload. In addition a function MUST NOT have overloads that differ only in the "type family" of a parameter. Type families are

- Integer: Edm.SByte, Edm.Byte, Edm.Int16, Edm.Int32, Edm.Int64, Edm.Decimal with Scale equal to zero
- Float: Edm.Single, Edm.Double, Edm.Decimal with Scale not equal to zero.

Attribute Name

12.2.1 Attribute Name

A function MUST provide a SimpleIdentifier value for the Name attribute. The combination of the function name and the unordered list of parameter names and types MUST be unique within its namespace.

12.2.2 Attribute ReturnType

If the return type is written with attribute notation, a

TypeName value MUST be provided for the ReturnType attribute.

If a value is provided for the return type attribute, the edm:Function element MUST NOT contain an edm:ReturnType element.

12.2.3 Attribute IsBindable

A function element MAY specify a Boolean value for the IsBindable attribute. If no value is specified for the IsBindable attribute, the value defaults to false.

If the value of the IsBindable attribute is set to true, the function element MUST contain at least one edm:Parameter element, and the first parameter is the binding parameter. It may be of any type.

12.2.4 Attribute IsComposable

A function element can specify a Boolean value for the IsComposable attribute. If no value is specified for the IsComposable attribute, the value defaults to false.

12.2.5 Attribute EntitySetPath

Bindable functions MAY specify a value for the EntitySetPath attribute if determination of the entity set for the return type is contingent on the binding parameter.

The value for the EntitySetPath attribute consists of a series of segments joined together with forward slashes.

The first segment of the entity set path MUST be the name of the binding parameter. The remaining segments of the entity set path MUST represent navigation segments or type casts.

A navigation segment names the SimpleIdentifier of the navigation property to be traversed. A type cast segment names the QualifiedName of the entity type that should be returned from the type cast.

12.3 Element edm:ReturnType

If the return type is written with element notation, the function element MUST contain a single `edm:ReturnType` element.

If element notation is used, the facet attributes ,`MaxLength`, `Precision`, `Scale`, and `SRID` can be used to specify the return type of the function. If the facet attributes are not specified, their values are considered unspecified.

12.3.1 Attribute Type

The `Type` attribute corresponds to the `ReturnType` attribute of the function element.

12.4 Element edm:Parameter

The `edm:Parameter` element allows one or more parameters to be passed to the function. This enables the function to return a dynamic set of instances – for example, the top-selling products by year. In this case the year must be specified as a parameter to the function with the `edm:Parameter` element.

12.4.1 Attribute Name

A parameter MUST provide a SimpleIdentifier value for the `Name` attribute. The parameter name MUST be unique within its parent element.

12.4.2 Attribute Type

A parameter MUST indicate which set of types can be passed to the parameter by providing a TypeName value for the `Type` attribute.

12.4.3 Parameter Facets

An `edm:Parameter` element MAY specify values for the `Nullable`, `DefaultValue`, `MaxLength`, `Precision`, `Scale`, or `SRID` attributes. The descriptions of these facets and their implications are covered in section 6.2.

13 Entity Container

An entity model can also describe how entities are logically grouped and even model the store or stores from which the entities can be retrieved. This is achieved through the declaration of entity containers and entity sets.

An entity set is a nominal type that allows access to entity type instances. Simple entity models frequently have one entity set per entity type, for example:

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

Other entity models may expose multiple entity sets per type. For instance, an entity model may have the following entity sets:

```
<EntitySet Name="Products" EntityType="Self.Product"/>
<EntitySet Name="DiscontinuedProducts" EntityType="Self.Product"/>
```

In this case the `Products` entity set could expose products that have not been discontinued and the `DiscontinuedProducts` entity set could expose products that have been discontinued. Note that an entity can be a member of at most one entity set, see **[OData-Protocol]**.

An entity set can expose instances of the specified entity type as well as any entity type inherited from the specified entity type.

A named entity allows addressing a single entity directly from the entity container without having to know its key. This allows defining a shortcut to “important” entities, or having “singleton” entities without the need for a one-element entity set.

A function import is used to expose functions that are defined in a data store. For example, the following function import exposes a stored procedure that returns the top ten revenue generating products for a given fiscal year:

```
<FunctionImport Name=" TopTenProductsByRevenue"
                Function="Self.TopTenProductsByRevenue"
                EntitySet="Products"/>
```

An entity container aggregates entity sets, root entities, and function imports.

A full example of an entity container is as follows:

```
<EntityContainer Name="DemoService">
 <EntitySet Name="Products" EntityType="Self.Product">
  <NavigationPropertyBinding Path="Category"
                             EntitySet="Self.DemoService.Categories"/>
  <NavigationPropertyBinding Path="Supplier"
                             EntitySet="Self.DemoService.Suppliers"/>
 </EntitySet>
 <EntitySet Name="Categories" EntityType="Self.Category">
  <NavigationPropertyBinding Path="Products"
                             EntitySet="Self.DemoService.Products"/>
 </EntitySet>
 <EntitySet Name="Suppliers" EntityType="Self.Supplier">
  <NavigationPropertyBinding Path="Products"
                             EntitySet="Self.DemoService.Products"/>
 </EntitySet>
 <Entity Name="Contoso" Type="Self.Supplier"/>
 <FunctionImport Name="ProductsByRating" Function="Self.ProductsByRating"
                 EntitySet="Products"/>
</EntityContainer>
```

13.1 Element edm:EntityContainer

The edm:EntityContainer element represents an entity container in an entity model. It corresponds to a logical data store and contains zero or more edm:EntitySet, edm:Entity, edm:ActionImport, or edm:FunctionImport elements. Action import, function import, entity set, and entity names MUST be unique within an entity container.

13.1.1 Attribute Name

The entity container MUST provide a unique SimpleIdentifier value for the Name attribute.

13.1.2 Attribute IsDefaultEntityContainer

The entity container MAY provide a Boolean value for the IsDefaultEntityContainer attribute. Each metadata document that is used to describe an OData service MUST mark exactly one entity container with this attribute to denote that it is the default.

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

13.1.3 Attribute Extends

An entity container MAY provide a QualifiedName value for the Extends attribute. The value provided to the Extends attribute MUST resolve to an entity container in scope. All children of the “base” entity container specified in the Extends attribute are added to the “extending” entity container that has the Extends attribute.

Example:

```
<EntityContainer Name="Extending" Extends="SomeOtherSchema.Base">
   ...
</EntityContainer>
```

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.

13.2 Element edm:EntitySet

The edm:EntitySet element is a nominal type that represents an entity set in an entity model.

13.2.1 Attribute Name

An entity set MUST provide a SimpleIdentifier value for the Name attribute.

13.2.2 Attribute EntityType

An entity set also has an EntityType attribute that MUST be provided with the QualifiedName of an entity type in scope. Each entity type in the model may have zero or more entity sets that reference the entity type.

An entity set MUST contain only instances of the entity type specified by the EntityType attribute or its subtypes. The entity type named by the entity type attribute MAY be abstract.

13.2.3 Attribute IncludeInServiceDocument

An entity set MAY provide a Boolean value for the IncludeInServiceDocument attribute. Its value indicates whether the entity set is advertised in the service document.

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

Entity sets that cannot be queried without specifying e.g. a `$filter` query option SHOULD specify the value `false` for this attribute.

13.3 Element `edm:NavigationPropertyBinding`

An entity set SHOULD contain an `edm:NavigationPropertyBinding` element for each navigation property of its entity type and each complex type used in its properties.

13.3.1 Attribute `Path`

A navigation property binding MUST name a navigation property of the entity set's entity type or one of its subtypes in the `Path` attribute. If the navigation property is defined on a subtype, the path attribute MUST contain the QualifiedName of the entity type, 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.

A navigation property MUST NOT be named in more than one navigation property binding; navigation property bindings are only used when all related entities are known to come from a single entity set.

13.3.2 Attribute `EntitySet`

A navigation property binding MUST specify a value for the `EntitySet` attribute. The value MUST be the name of the entity set that contains the related instances targeted by the navigation property specified in the `Path` attribute. If the target entity set is not defined in the same entity container as the enclosing `EntitySet` element, the entity set name MUST be qualified with the namespace or alias of the schema that defines the entity set, followed by the entity container.

Examples:

- `EntitySet="SomeSet"` for an entity set in the same container as the enclosing entity set,
- `EntitySet="SomeModel.SomeContainer.SomeSet"` for an entity set in any container in scope.

13.4 Element `edm:Entity`

The `edm:Entity` element represents a single entity in an entity model.

13.4.1 Attribute `Name`

A named entity MUST provide a SimpleIdentifier value for the `Name` attribute.

13.4.2 Attribute `Type`

A named entity also has a `Type` attribute that MUST be provided with the QualifiedName of an entity type in scope. Each entity type in the model may have zero or more named entities that reference the entity type.

A named entity MUST reference an instance of the entity type specified by the `Type` attribute or its subtypes. The entity type named by the `Type` attribute MAY be abstract.

13.5 Element `edm:ActionImport`

The `edm:ActionImport` element allows exposing an Action as a top-level element in an entity container or bind the action result to a specific entity set. Action imports are never advertised in the service document,

13.5.1 Attribute Name

An action import MUST provide a SimpleIdentifier value for the Name attribute. It MAY be identical to the last SimpleIdentifer segment of the Action attribute value.

13.5.2 Attribute Action

An action import MUST provide a QualifiedName value for the Action attribute which MUST resolve to the name of an edm:Action element in scope.

13.5.3 Attribute EntitySet

If the return type of the action specified in the Action attribute is an entity or a collection of entities, a SimpleIdentifier or QualifiedName value MAY be defined for the EntitySet attribute that names the entity set to which the returned entities belong. If a SimpleIdentifier is specifid, it MUST resolve to an entity set defined in the same entity container. If a QualifiedName is specified, it MUST resolve to an entity set in scope.

If the return type is not an entity or a collection of entities, a value MUST NOT be defined for the EntitySet attribute.

If the EntitySet attribute is assigned a value, it overrides the EntitySetPath attribute of the action specified in the Action attribute.

13.6 Element edm:FunctionImport

The edm:FunctionImport element allows exposing a Function as a top-level element in an entity container or bind the function result to a specific entity set.

13.6.1 Attribute Name

A function import MUST provide a SimpleIdentifier value for the Name attribute. It MAY be identical to the last SimpleIdentifer segment of the Function attribute value.

13.6.2 Attribute Function

A function import MUST provide a QualifiedName value for the Function attribute which MUST resolve to the name of an edm:Function element in scope.

13.6.3 Attribute EntitySet

If the return type of the function specified in the Function attribute is an entity or a collection of entities, a SimpleIdentifier or QualifiedName value MAY be defined for the EntitySet attribute that names the entity set to which the returned entities belong. If a SimpleIdentifier is specified, it MUST resolve to an entity set defined in the same entity container. If a QualifiedName is specified, it MUST resolve to an entity set in scope.

If the return type is not an entity or a collection of entities, a value MUST NOT be defined for the EntitySet attribute.

If the EntitySet attribute is assigned a value, it overrides the EntitySetPath attribute of the function specified in the Function attribute.

13.6.4 Attribute `IncludeInServiceDocument`

A function import for a parameterless function MAY provide a Boolean value for the `IncludeInServiceDocument` attribute. Its value indicates whether the function import is advertised in the service document.

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

14 Vocabulary and Annotation

The concept of “Vocabularies and Annotations” provides the ability to annotate metadata as well as instance data, and define a powerful extensibility point for OData. An *annotation* attaches a *term* to a model element and provides a means of calculating a value for the 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 should override the annotation specified at the metadata level. Instance annotations appear in the actual payload as described in **[OData-Atom]** and **[OData-JSON]**. Annotations that apply across instances should be specified as metadata annotations.

A *vocabulary* is a namespace containing a set of terms where each term is a named metadata extension. Anyone can define a vocabulary (a set of terms) that is scenario-specific or company-specific; more commonly used terms can be published as shared vocabularies such as the OData core vocabulary.

An annotated term can be used for two fundamental purposes:

- To extend model elements and type instances with additional information.
- To map instances of annotated entity types to an interface; the interface is defined by the term type.

A service SHOULD NOT require a client to interpret annotations.

Example: extend an entity type with a `DisplayName` by a metadata annotation that binds the term `DisplayName` to the value of the property `Name`:

```
<EntityType Name="Category">
 ...
 <Property Name="Name" Nullable="true" Type="Edm.String"/>
 <Annotation Term="UI.DisplayName" Path="Name"/>
</EntityType>
```

Annotations also allow viewing instances of a structured type as instances of a differently structured type specified by the applied term. For instance, the following `Product` entity type includes an annotation that allows its instances to be viewed as instances of the type specified by the term `SearchResult`. For instance, the following `Product` entity type includes an annotation that allows its instances to be viewed as instances of the complex type `SearchResult`:

```
<EntityType Name="Product">
 <Key>
  <PropertyRef Name="ID"/>
 </Key>
 <Property Name="ID" Nullable="false" Type="Edm.Int32"/>
 <Property Name="Name" Nullable="true" Type="Edm.String"/>
 <Property Name="Description" Nullable="true" Type="Edm.String" />
 <Property Name="ReleaseDate" Nullable="false" Type="Edm.DateTimeOffset"/>
 <Property Name="Rating" Nullable="false" Type="Edm.Int32"/>
 <Property Name="Price" Nullable="false" Type="Edm.Decimal"/>
 ...
 <Annotation Term="SearchVocabulary.SearchResult">
  <PropertyValue Property="Title" Path="Name"/>
  <PropertyValue Property="Url">
   <Apply Function="odata.concat">
    <String>Products(</String>
    <Path>ID</Path>
    <String>)</String>
   </Apply>
  </PropertyValue>
  <PropertyValue Property="Abstract">
   <Path>Description</Path>
  </PropertyValue>
 </Annotation>
</EntityType>
```

14.1 Element `Term`

The `edm:Term` element defines a term in a vocabulary.

A term is defined to annotate a CSDL element with additional data. A term has a value that MAY be of primitive type, enumeration type, complex type, entity type, or a collection of these types.

14.1.1 Attribute `Name`

The `edm:Term` element MUST provide a SimpleIdentifier value for the `Name` attribute. The `Name` attribute allows the term to be applied with an annotation.

14.1.2 Attribute `Type`

The `edm:Term` element MUST provide a TypeName value for the `Type` attribute. The `Type` attribute indicates what type of value must be returned by the expression contained in the annotation.

14.1.3 Attribute `DefaultValue`

A `edm:Term` element of primitive or enumeration type MAY define a value for the `DefaultValue` attribute. The value of this attribute determines the value of the term when applied in an `edm:Annotation` without providing an expression.

Default values MUST be represented according to the `xxxBody` rule defined in **[OData-ABNF]** that is appropriate for the type of the property.

If no value is specified, the `DefaultValue` attribute defaults to `null`.

14.1.4 Attribute `AppliesTo`

A `edm:Term` element MAY define a value for the `AppliesTo` attribute. The value of this attribute is a whitespace-separated list of CSDL element names that this term can be applied to. If no value is supplied, the term is not restricted in its application.

Example:

```
<Term Name="IsURI" Type="Core.Tag"Term"DefaultValue="true"
      AppliesTo="Property">
  <Annotation Term="Core.Description">
    <String>
      Properties and terms annotated with this term MUST contain a valid URI
    </String>
  </Annotation>
  <Annotation Term="Core.RequiresType" String="Edm.String"/>
</Term>
```

The `IsURI` term can be applied to properties and terms that are of type `Edm.String`.

14.1.5 Term Facets

An `edm:Term` element MAY specify values for the `Nullable`, `DefaultValue`, `MaxLength`, `Precision`, `Scale`, or `SRID` attributes. The descriptions of these facets and their implications are covered in section 6.2.

14.2 Element `edm:Annotations`

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

14.2.1 Attribute `Target`

An `edm:Annotations` element MUST assign a path expression to the `Target` attribute. The value of the `Target` attribute MUST resolve to a model element in the entity model.

An annotations element contains zero or more `edm:Annotation` elements.

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. For instance, the following annotation hints that it 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* attaches a *term* to a model element and provides a means of calculating a value for the term. The following model elements MAY be annotated with a term:

- `edm:Action`
- `edm:ActionImport`
- `edm:Annotations`
- `edm:Apply`
- `edm:AssertType`
- `edm:Collection`
- `edm:ComplexType`
- `edm:Entity`
- `edm:EntityContainer`

- edm:EntitySet
- edm:EntityType
- edm:EnumType
- edm:Function
- edm:FunctionImport
- edm:If
- edm:IsType
- edm:LabeledElement
- edm:Member
- edm:NavigationProperty
- edm:Null
- edm:OnDelete
- edm:Parameter
- edm:Property
- edm:PropertyValue
- edm:Record
- edm:ReferentialConstraint
- edm:Schema
- edm:Term
- edm:TypeDefinition
- edmx:Reference

An annotation element MUST be used as a child of the model element it annotates or as a child of an edm:Annotations element that targets the appropriate model element.

An annotation element MAY contain a constant expression or dynamic expression in either attribute or element notation. If no expression is specified, the default value of the term definition is used.

If an entity type or complex type is annotated with a term that itself has a structured type, an instance of the type may be viewed as an "instance" of the term, and the term Name may be used as a "term cast" segment in path expressions.

14.3.1 Attribute Term

An annotation element MUST provide a QualifiedName value for the Term attribute. The value of the Term attribute MUST be the Name of a Term definition 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: The following annotation hints that it 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

Values for a term or properties of a term are obtained by calculating expressions. There are a variety of expressions that allow service authors to supply constant values.

The following examples show 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>
```

The constant expressions and the `edm:NavigationPropertyPath`, `edm:Path`, `edm:PropertyPath`, and `edm:Url` dynamic expressions also support attribute notation:

```
Expression edm:Binary
```

The `edm:Binary` expression evaluates to a primitive binary value. A binary expression MUST be assigned a value of type `xs:hexBinary`, see **[XML-Schema-2]**, section 3.2.15.

The binary expression MAY be provided using element notation or attribute notation:

```
<Annotation Term="org.example.display.Thumbnail" Binary="3f3c6d78206c"/>

<Annotation Term="org.example.display.Thumbnail">
 <Binary>3f3c6d78206c</Binary>
</Annotation>
```

14.4.1 Expression `edm:Bool`

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

The Boolean expression MAY be provided using element notation or attribute notation:

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

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

14.4.2 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 NOT contain a time-zone offset.

The date expression MAY be provided using element notation or attribute notation:

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

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

14.4.3 Expression edm:DateTimeOffset

The edm:DateTimeOffset expression evaluates to a primitive date/time value with a time-zone offset. A date/time expression MUST be assigned a value of type xs:dateTimeStamp, see **[XML-Schema-2]**, section 3.4.28. The value MUST NOT contain an end-of-day fragment (24:00:00).

The date/time expression MAY be provided using element notation or attribute notation:

```
<Annotation Term="org.example.display.LastUpdated"
            DateTimeOffset="2000-01-01T16:00:00.000Z"/>

<Annotation Term="org.example.display.LastUpdated">
 <DateTimeOffset>2000-01-01T16:00:00.000-09:00</DateTimeOffset>
</Annotation>
```

14.4.4 Expression edm:Decimal

The edm:Decimal expression evaluates to a primitive decimal value. A decimal expression MUST be assigned a value of the type xs:decimal, see **[XML-Schema-2]**, section 3.2.3.

The decimal expression MAY be provided using element notation or attribute notation:

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

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

14.4.5 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:

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

<Annotation Term="org.example.task.duration">
 <Duration>P11D23H59M59.999999999999S</Duration>
</Annotation>
```

14.4.6 Expression edm:EnumMember

The edm:EnumMember expression enables a value to be obtained by referencing a member of an enumeration type. An enumeration member expression MUST be assigned a value that consists of the qualified name of the enumeration type, followed by a dot 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:

```
 <Annotation Term="org.example.HasPattern"
             EnumMember="org.example.Pattern.Yellow"/>

<Annotation Term="org.example.HasPattern">
 <EnumMember>
  org.example.Pattern.Yellow
  org.example.Pattern.Striped
 </EnumMember>
</Annotation>
```

14.4.7 Expression edm:Float

The edm:Float expression evaluates to a primitive floating point (or double) value. A float expression MUST be assigned a value of the type xs:double, see **[XML-Schema-2]**, section 3.2.5.

The float expression MAY be provided using element notation or attribute notation:

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

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

14.4.8 Expression edm:Guid

The edm:Guid expression evaluates to a primitive 32-character string value. A guid expression MUST be assigned a value conforming to the rule guid in **[OData-ABNF]**.

The guid expression MAY be provided using element notation or attribute notation:

```
<Annotation Term="org.example.display.Id"
            Guid="21EC2020-3AEA-1069-A2DD-08002B30309D"/>

<Annotation Term="org.example.display.Id">
 <Guid>21EC2020-3AEA-1069-A2DD-08002B30309D</Guid>
</Annotation>
```

14.4.9 Expression edm:Int

The edm:Int expression evaluates to a primitive integer value. An integer MUST be assigned a value of the type xs:integer, see **[XML-Schema-2]**, section 3.3.13.

The integer expression MAY be provided using element notation or attribute notation:

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

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

14.4.10 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 3.2.1.

The string expression MAY be provided using element notation or attribute notation:

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

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

14.4.11 Expression edm:TimeOfDay

The edm:TimeOfDay expression evaluates to a primitive time value. On platforms that do not support a primitive time value, the time expression evaluates to a primitive date/time value. A TimeOfDay expression MUST be assigned a value of the type xs:time see **[XML-Schema-2]**, section 3.3.8. The value MUST NOT contain an end-of-day fragment (24:00:00) or a time-zone offset.

The time expression MAY be provided using element notation or attribute notation:

```
<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

Values for a term or properties of a term are obtained by calculating expressions. There are a variety of expressions that allow service authors to supply dynamic values.

14.5.1 Expression `edm:Apply`

The `edm:Apply` expression enables a value to be obtained by applying a client-side function.

The `Apply` expression contains or more expressions. The expressions contained within the `Apply` expression are used as parameters to the function.

The edm:`Apply` expression MUST be written with element notation.

14.5.1.1 Attribute `Function`

An `Apply` expression MUST assign a QualifiedName value to the `Function` attribute. The value of the function attribute is used to locate the client-side function that should be applied.

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.

14.5.1.1.1 Function `odata.concat`

The `odata.concat` standard client-side function takes two or more expressions as arguments. Each argument MUST evaluate to a primitive or enumeration type. It returns a value of type `Edm.String` that is the concatenation of the literal representations of the results of the argument expressions; see **[OData-ABNF]**.

Example:

```
<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>
```

Here `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.

14.5.1.1.2 Function `odata.fillUriTemplate`

The `odata.fillUriTemplate` standard client-side function takes two or more expressions as arguments.

The first argument MUST be of type `Edm.String` and specifies a URI template according to **[RFC6570]**, the other arguments MUST be `edm:LabeledElement` expressions. Each `edm:LabeledElement`

expression specifies the template parameter name in its `Name` attribute and evaluates to the template parameter value.

[RFC6570] defines three kinds of template parameters: simple values, lists of values, and key-value maps.

Simple values are represented as `edm:LabeledElement` expressions that evaluate to a single primitive value. The literal representation of this value according to **[OData-ABNF]** is used to fill the corresponding template parameter.

Lists of values are represented as `edm:LabeledElement` expressions that evaluate to a collection of primitive values.

Key-value maps are represented as `edm:LabeledElement` expressions that evaluate to a collection of complex types with two properties that are used in lexicographic order. The first property is used as key, the second property as value.

Example assuming there are no special characters in values of the `NameOfMovieGenre` property:

```
<Apply Function="odata.fillUriTemplate">
  <String>http://host/service/Genres('{genreName}')</String>
  <LabeledElement Name="genreName" Path="NameOfMovieGenre" />
</Apply>
```

14.5.1.1.3 Function `odata.uriEncode`

The `odata.uriEncode` standard client-side function takes one argument of primitive type and returns the URL-encoded OData literal that can be used as a key value in OData URLs or in the query part of OData URLs. Note that string literals are surrounded by single quotes.

Example:

```
<Apply Function="odata.fillUriTemplate">
  <String>http://host/service/Genres({genreName})</String>
  <LabeledElement Name="genreName">
    <Apply Function="odata.uriEncode" >
      <Path>NameOfMovieGenre</Path>
    </Apply>
  </LabeledElement>
</Apply>
```

14.5.2 Expression `edm:AssertType`

The `edm:AssertType` expression asserts that a value obtained from a child expression is of a specified type. The value calculated by the assert type expression is the value obtained from the child expression casted to the specified type.

The assert-type expression MUST specify a `Type` attribute contain exactly one expression. The expression contained within the assert type expression is used as a parameter to the type assertion.

The assert-type expression MUST be written with element notation:

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

14.5.2.1 Attribute `Type`

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

14.5.3 Expression edm:Collection

The edm:Collection expression enables a value to be obtained from zero or more child expressions. The value calculated by the collection expression is the collection of the values calculated by each of the child expressions.

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

A collection expression MUST be written with element notation:

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

14.5.4 Expression edm:If

The edm:If expression enables a value to be obtained by evaluating a conditional expression. It MUST contain exactly three child elements with dynamic or static expressions.

The first child expression is the conditional expression and MUST evaluate to a Boolean result.

The second and third child expressions are the expressions which are evaluated conditionally. They MUST return type compatible values.

If the first expression evaluates to true, the second child expression MUST be evaluated and its value MUST be returned as the result of the edm:If expression. If the conditional expression evaluates to false, the third child expression MUST be evaluated and its value MUST be returned as the result of the edm:If expression.

The edm:If expression MUST be written with element notation, as shown in the following example:

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

14.5.5 Expression edm:IsType

The edm:IsType expression evaluates a child expression and returns a Boolean value indicating whether the child expression returns the specified type.

An edm:IsType expression MUST specify a Type attribute and contain exactly one child expression. The edm:IsType expression MUST return true if the child expression returns a type that is compatible with the type named in the Type attribute. The edm:IsType 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:IsType expression MUST be written with element notation:

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

14.5.5.1 Attribute `Type`

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

14.5.6 Expression `edm:LabeledElement`

The `edm:LabeledElement` expression assigns a name to a child expression. The value of the child expression can then be reused elsewhere with an `edm:LabeledElementReference` expression.

A labeled-element expression MUST contain exactly one child expression written either in attribute notation or element notation. The value of the child expression is passed through the labeled-element expression.

A labeled-element expression MUST be written with element notation:

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

14.5.6.1 Attribute `Name`

An `edm:LabeledElement` expression MUST provide a SimpleIdentifier value for the `Name` attribute.

14.5.7 Expression `edm:LabeledElementReference`

The `edm:LabeledElementReference` expression returns the value of an `edm:LabeledElement` expression.

The labeled-element reference expression MUST contain the SimpleIdentifier name of a labeled element expression in scope.

The labeled-element reference expression MUST be written with element notation:

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

14.5.8 Expression `edm:Null`

The `edm:Null` expression returns an untyped null value. The null expression MUST NOT contain any other elements or expressions.

The null expression MUST be written with element notation:

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

14.5.9 Expression `edm:NavigationPropertyPath`

The `edm:NavigationPropertyPath` expression provides a value for terms or term properties that specify the built-in abstract type `Edm.NavigationPropertyPath`. It uses the same syntax and rules as the `edm:Path` expression, with the added restriction that the last path segment MUST resolve to a navigation property in the context of the preceding path part.

In contrast to the `edm:Path` expression the value of the `edm:NavigationPropertyPath` expression is the path itself, not the target instance(s) of the navigation property identified by the path. This is useful for terms that describe the semantics of a group of navigation properties and thus cannot be applied to a single navigation property.

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

```
<Annotation Term="UI.HyperLink" NavigationPropertyPath="Supplier"/>

<Annotation Term="Capabilities.UpdateRestrictions">
  <PropertyValue Property="NonUpdatableNavigationProperties">
    <Collection>
      <NavigationPropertyPath>Supplier</NavigationPropertyPath>
      <NavigationPropertyPath>Category</NavigationPropertyPath>
    </Collection>
  </PropertyValue>
</Annotation>
```

14.5.10 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 entity types, and properties of entity types and complex types.

The value assigned to the path expression MUST be composed of zero or more path segments joined together by forward slashes (/).

If the 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 a null value.

If the path segment start 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 annotations are implicitly “annotated” for media entities and named stream properties:

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

If the 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.

If a path segment is the name of a navigation property that has a cardinality of many, the path MUST NOT have any subsequent segments other than at most one type cast, term cast, or a $count segment. If the last segment is a $count segment, the path evaluates to the number of related entities.

Annotations MAY be embedded within their target, or embedded within an edm:Annotations element that specifies the annotation target with a path expression in its Target attribute. The latter situation is referred to as *targeting* in the remainder of this section.

For annotations embedded within or targeting an entity container the path expression is evaluated starting at the entity container, i.e. an empty path resolves to the entity container, and non-empty path values MUST start with the name of a container child (entity set, function import, action import, or named entity). 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 named entity the path expression is evaluated starting at the entity set, i.e. an empty path resolves to the entity set, and non-empty paths MUST follow the rules for annotations targeting the declared entity type of the entity set or named entity.

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 in the path expression of the `Target` attribute of the enclosing `edm:Annotations` element, i.e. an empty path resolves to the outermost type, and the first segment of a non-empty path MUST be a property or navigation property of the outermost type, a type cast, or a term cast.

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

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

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

14.5.11 Expression `edm:PropertyPath`

The `edm:PropertyPath` expression provides a value for terms or term properties that specify the built-in abstract type `Edm.PropertyPath`. It uses the same syntax and rules as the `edm:Path` expression, with the added restriction that the last path segment MUST resolve to a property in the context of the preceding path part. It MUST NOT resolve to a navigation property.

In contrast to the `edm:Path` expression the value of the `edm:PropertyPath` expression is the path itself, not the value of the property identified by the path. This is useful for terms that describe the semantics of a group of properties and thus cannot be applied to a single property.

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

```
<Annotation Term="UI.RefreshOnChangeOf" PropertyPath="ChangedAt"/>

<Annotation Term="Capabilities.UpdateRestrictions">
  <PropertyValue Property="NonUpdatableProperties">
    <Collection>
      <PropertyPath>CreatedAt</PropertyPath>
      <PropertyPath>ChangedAt</PropertyPath>
    </Collection>
  </PropertyValue>
</Annotation>
```

14.5.12 Expression `edm:Record`

The `edm:Record` expression enables a new entity type or complex type instance to be constructed.

A record expression contains zero or more `edm:PropertyValue` elements. For each non-nullable property of the record construct's type an `edm:PropertyValue` child element MUST be provided. For derived types this rule applies only to properties directly defined by the derived type.

A record expression MUST be written with element notation, as shown in the following example:

```
<Annotation Term="org.example.person.Employee">
 <Record>
  <PropertyValue Property="GivenName" Path="FirstName"/>
  <PropertyValue Property="Surname" Path="LastName"/>
 </Record>
</Annotation>
```

14.5.12.1 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.

14.5.12.2 Element `edm:PropertyValue`

The `edm:PropertyValue` element supplies a value to a property on the type instantiated by an `edm:Record` expression. The value is obtained by evaluating an expression.

The `PropertyValue` element MUST contain exactly one expression. The expression MAY be provided using element notation or attribute notation.

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.

14.5.13 Expression `edm:Url`

The `edm:Url` expression enables a value to be obtained by sending a `GET` request to the value of the `Url` expression.

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

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

```
<Annotation Term="Vocab.Supplier">
 <Url>
  <Apply Function="odata.fillUriTemplate">
   <String>http://host/service/Suppliers({suppID})</String>
   <LabeledElement Name="suppID">
    <Apply Function="odata.uriEncode">
     <Path>SupplierId</Path>
    </Apply>
   </LabeledElement>
  </Apply>
 </Url>
</Annotation>
```

15 Metadata Service Schema

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

With `~` as an abbreviation for the service root URL, the Metadata Service root URL is `~/$metadata/`, and the request

```
GET ~/$metadata/$metadata
```

MUST return the CSDL document **[OData-Meta]**.

Schemas are identified with a `Fullname` key property that is the alias of the schema, and if no alias is defined, the namespace of the schema, for example:

```
GET ~/$metadata/Schemata('SampleModel')
```

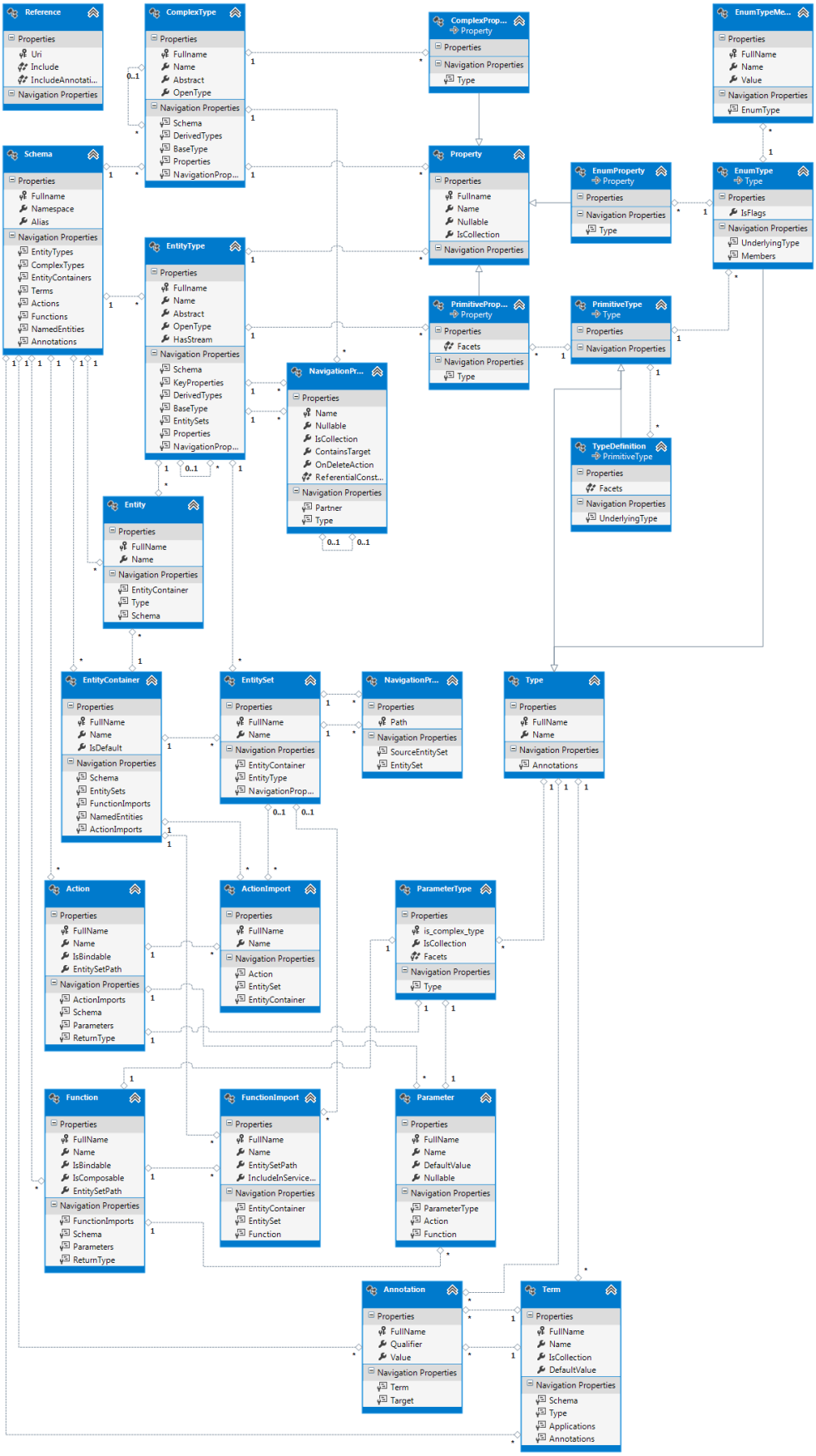
Model elements that are identified with a `Name` attribute within their parent element are represented in this schema by entity types in with a `Fullname` key property whose value is obtained by appending a dot (`.`) and the `Name` of the model element to the `Fullname` of the entity representing the parent element, for example:

```
GET ~/$metadata/EntityTypes('SampleModel.Customer')
GET ~/$metadata/Properties('SampleModel.Customer.ID')
GET ~/$metadata/EntitySets('SampleModel.SalesData.Customers)
```

Annotations are identified by the combination of their target, term, and qualifier. The Fullname value is constructed by appending the `Fullname` of the target with an at (`@`) sign and the `Fullname` of the term, and for non-empty qualifiers with a hash (`#`) sign and the qualifier, for example:

```
GET ~/$metadata/Annotations('SampleModel.Customer@UI.DisplayName#Tablet')
```

The following graphical representation of this schema is incomplete. For example all model elements that can be annotated have a navigation property Annotations to the Annotation entity type but they are only depicted in `Schema`, `Type`, and `Term` to keep the diagram legible:



16 CSDL Examples

Following are two basic examples of valid EDM models as represented in CSDL. These examples demonstrate many of the topics covered above.

16.1 Products and Categories Example

```
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx"
           Version="4.0">
  <edmx:Reference Uri="http://tinyurl.com/Org-OData-Measures-V1">
    <edmx:Include Alias="UoM" Namespace="Org.OData.Measures.V1" />
  </edmx:Reference>
  <edmx:DataServices>
    <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm"
            Namespace="ODataDemo">
      <EntityType Name="Product" HasStream="true">
        <Key>
          <PropertyRef Name="ID"/>
        </Key>
        <Property Name="ID" Type="Edm.Int32" Nullable="false"/>
        <Property Name="Name" Type="Edm.String" Nullable="true"/>
        <Property Name="Description" Type="Edm.String" Nullable="true"/>
        <Property Name="ReleaseDate" Type="Edm.Date" Nullable="false"/>
        <Property Name="DiscontinuedDate" Type="Edm.Date" Nullable="true"/>
        <Property Name="Rating" Type="Edm.Int32" Nullable="false"/>
        <Property Name="Price" Type="Edm.Decimal" Nullable="false">
          <Annotation Term="UoM.ISOCurrency" Path="Currency"/>
        </Property>
        <Property Name="Currency" Type="Edm.String" Nullable="false"
                  MaxLength="3"/>
        <NavigationProperty Name="Category" Type="ODataDemo.Category"
                            Nullable="false" Partner="Products">
          <OnDelete Action="Cascade"/>
        </NavigationProperty>
        <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" Nullable="true"/>
        <NavigationProperty Name="Products" Partner="Category"
                            Type="Collection(ODataDemo.Product)"/>
      </EntityType>
      <EntityType Name="Supplier">
        <Key>
          <PropertyRef Name="ID"/>
        </Key>
        <Property Name="ID" Type="Edm.Int32" Nullable="false"/>
        <Property Name="Name" Type="Edm.String" Nullable="true"/>
        <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)"/>
        <NavigationProperty Name="Country" Type="ODataDemo.Country">
          <ReferentialConstraint Property="Address/Country"
                                 ReferencedProperty="Name"/>
        </NavigationProperty>
```

```
      </EntityType>
      <EntityType Name="Country">
        <Key>
          <PropertyRef Name="Name"/>
        </Key>
        <Property Name="Name" Type="Edm.String"/>
      </EntityType>
      <ComplexType Name="Address">
        <Property Name="Street" Type="Edm.String" Nullable="true"/>
        <Property Name="City" Type="Edm.String" Nullable="true"/>
        <Property Name="State" Type="Edm.String" Nullable="true"/>
        <Property Name="ZipCode" Type="Edm.String" Nullable="true"/>
        <Property Name="Country" Type="Edm.String" Nullable="true"/>
      </ComplexType>
      <Function Name="ProductsByRating"
                ReturnType="Collection(ODataDemo.Product)">
        <Parameter Name="Rating" Type="Edm.Int32" DefaultValue="4"/>
      </Function>
      <EntityContainer Name="DemoService" IsDefaultEntityContainer="true">
        <EntitySet Name="Products" EntityType="ODataDemo.Product">
          <NavigationPropertyBinding Path="Category" EntitySet="Categories"/>
          <NavigationPropertyBinding Path="Supplier" EntitySet="Suppliers"/>
        </EntitySet>
        <EntitySet Name="Categories" EntityType="ODataDemo.Category">
          <NavigationPropertyBinding Path="Products" EntitySet="Products"/>
        </EntitySet>
        <EntitySet Name="Suppliers" EntityType="ODataDemo.Supplier">
          <NavigationPropertyBinding Path="Products" EntitySet="Products"/>
          <Annotation Term="Core.OptimisticConcurrencyControl">
            <Record>
              <PropertyValue Property="ETagDependsOn">
                <Collection>
                  <PropertyPath>Concurrency</PropertyPath>
                </Collection>
              </PropertyValue>
            </Record>
          </Annotation>        </EntitySet>
        <Entity Name="Contoso" Type="Self.Supplier"/>
        <EntitySet Name="Countries" EntityType="ODataDemo.Country"/>
        <FunctionImport Name="ProductsByRating" EntitySet="Products"
                        Function="ODataDemo.ProductsByRating"/>
      </EntityContainer>
    </Schema>
  </edmx:DataServices>
</edmx:Edmx>
```

16.2 Annotated Customers and Orders Example

```
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx"
           Version="4.0">
  <edmx:Reference Uri="http://tinyurl.com/Org-OData-Measures-V1">
    <edmx:Include Namespace="Org.OData.Measures.V1" />
  </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="Model1" Alias="Self">
      <EntityContainer Name="Model1Container" IsDefaultEntityContainer="true">
        <EntitySet Name="CustomerSet" EntityType="Model1.Customer"/>
        <EntitySet Name="OrderSet" EntityType="Model1.Order"/>
      </EntityContainer>
```

```
<Annotations Target="Self.Customer">
  <Annotation Term="Vocabulary1.EMail">
    <Null />
  </Annotation>
  <Annotation Term="AccountID" Path="AccountNumber"/>
  <Annotation Term="Title" String="Customer Info"/>
</Annotations>
<EntityType Name="Customer">
  <Key>
    <PropertyRef Name="CustomerId"/>
  </Key>
  <Property Name="CustomerId" Type="Edm.Int32" Nullable="false"/>
  <Property Name="FirstName" Type="Edm.String" Nullable="true"/>
  <Property Name="LastName" Type="Edm.String" Nullable="true"/>
  <Property Name="AccountNumber" Type="Edm.Int32" Nullable="true"/>
  <Property Name="Address" Type="Self.Address" Nullable="false"/>
  <NavigationProperty Name="Orders" Type="Collection(Self.Order)"/>
  <Annotation Term="Vocabulary1.Person">
    <Record>
      <PropertyValue Property="DisplayName">
        <Apply Function="odata.concat">
          <Path>FirstName</Path>
          <String> </String>
          <Path>LastName</Path>
        </Apply>
      </PropertyValue>
    <Record>
  </Annotation>
</EntityType>
<EntityType Name="Order">
  <Key>
    <PropertyRef Name="OrderId"/>
  </Key>
  <Property Name="OrderId" Type="Edm.Int32" Nullable="false"/>
  <Property Name="OrderDate" Type="Edm.Int32" Nullable="true"/>
  <Property Name="Description" Type="Edm.String" Nullable="true"/>
  <NavigationProperty Name="Customer" Type="Self.Customer"
                      Nullable="false"/>
  <NavigationProperty Name="Product" Type="Self.Product"
                      Nullable="false"/>
</EntityType>
<EntityType Name="SalesOrder" BaseType="Self.Order">
  <Property Name="Paid" Type="Edm.Boolean" Nullable="false"/>
</EntityType>
<EntityType OpenType="true" Name="Product">
  <Key>
    <PropertyRef Name="ProductId"/>
  </Key>
  <Property Name="ProductId" Type="Edm.Int32" Nullable="false"/>
  <Property Name="Name" Type="Edm.String" Nullable="false"/>
  <Property Name="Description" Type="Edm.String" Nullable="true"/>
  <Property Name="Size" Type="Self.Size" Nullable="true"/>
</EntityType>
<ComplexType Name="Address">
  <Property Name="Street" Type="Edm.String" Nullable="false"/>
  <Property Name="City" Type="Edm.String" Nullable="false"/>
  <Property Name="State" Type="Edm.String" Nullable="false"/>
  <Property Name="Zip" Type="Edm.String" Nullable="false"/>
  <Property Name="Position" Type="Edm.GeographyPoint" Nullable="false"
            SRID="4326"/>
</ComplexType>
<TypeDefinition Name="Length" UnderlyingType="Edm.Int32">
  <Annotation Term="Org.OData.Measures.V1.Unit" String="Centimeters"/>
</TypeDefinition>
```



```
      <TypeDefinition Name="Weight" UnderlyingType="Edm.String">
        <Annotation Term="Org.OData.Measures.V1.Unit"
                    String="Kilograms"/>
      </TypeDefinition>
      <ComplexType Name="Size">
        <Property Name="Width" Type="Self.Length"/>
        <Property Name="Depth" Type="Self.Length"/>
        <Property Name="Height" Type="Self.Length"/>
        <Property Name="Weight" Type="Self.Weight"/>
      </ComplexType>
    </Schema>
  </edmx:DataServices>
</edmx:Edmx>
```


17 Attribute Values

17.1 Namespace

A Namespace is a character sequence conforming to the rule `namespace` in **[OData-ABNF]**.Non-normatively speaking it is a dot-separated sequence of SimpleIdentifiers with a maximum length of 511 Unicode characters.

17.2 SimpleIdentifier

A SimpleIdentifier is a character sequence conforming to the rule `odataIdentifier` in **[OData-ABNF]**.

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

17.3 QualifiedName

For model elements that are direct children of a schema: the namespace or alias of the schema that defines the model element, followed by a dot and the name of the model element, see rule `qualifiedTypeName` in **[OData-ABNF]**.

For model elements that are direct children of an entity container: the qualified name of the entity container, followed by a dot and the name of the model element, see rules, `qualifiedActionName` and `qualifiedFunctionName` in **[OData-ABNF]**.

For built-in primitive types: the name of the type, prefixed with `Edm` followed by a dot.

17.4 TypeName

The QualifiedName of a built-in primitive or abstract type, a type definition, complex type, enumeration type, or entity type, or a collection of one of these types, see rule `qualifiedTypeName` in **[OData-ABNF]**.

The type must be in scope, i.e. the type MUST be defined in the `Edm` namespace or it MUST be defined in the schema identified by the namespace or alias portion of the qualified name, and the identified schema MUST be defined in the same CSDL document or included from a directly referenced document.

17.5 Boolean

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

18 Conformance

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

Conforming clients MUST be prepared to consume a model that uses any or all of the constructs defined in this specification, including custom annotations defined by the service, and MUST ignore any elements or attributes not defined in this version of the specification.

Appendix A. Acknowledgments

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

Appendix B. Revision History

Revision	Date	Editor	Changes Made
Working Draft 01	2012-08-22	Michael Pizzo	Translated Contribution to OASIS format/template
Committee Specification Draft 01	2013-04-26		Simplified annotations, relationships, added containment, named entities Added Type Definitions, Edm.Date, Edm.TimeOfDay, Edm.Duration datatypes. Retired Edm.DateTime, Edm.Time. Enhanced ComplexType support Expanded Service Document Fleshed out descriptions and examples and addressed numerous editorial and technical issues processed through the TC Added Conformance section