Model Functions?

CollectionType/TypeRef?

ReferenceType/RowType?

dataservice attributes (i.e., to functions)

Model Reference/Annotation Reference

MARK’S TODOS

Link up the ABNF somehow  
Consider consolidating sections into coarser granularity  
Ensure all examples use customers-orders-products motif

Table of Contents

[Appendix A: Formal Common Schema Definition Language (CSDL) 6](#_Toc320467234)

[1 Common Schema Definition Language (CSDL) Namespaces 6](#_Toc320467235)

[1.1 Entity Data Model for Data Services Packaging (EDMX) Namespace 6](#_Toc320467236)

[1.2 Entity Data Model (EDM) Namespace 6](#_Toc320467237)

[1.3 Data Service Metadata Namespace 7](#_Toc320467238)

[2 Common Characteristics of Entity Model Elements 7](#_Toc320467239)

[2.1 Nominal Elements 7](#_Toc320467240)

[2.2 Structured Elements 7](#_Toc320467241)

[2.3 The edm:Documentation Element 8](#_Toc320467242)

[2.4 Vocabulary Annotations 8](#_Toc320467243)

[2.5 Custom Annotations 8](#_Toc320467244)

[2.6 Primitive Types 9](#_Toc320467245)

[3 Entity Model Wrapper Constructs 10](#_Toc320467246)

[3.1 The edmx:Edmx Element 10](#_Toc320467247)

[3.1.1 The Version Attribute 11](#_Toc320467248)

[3.2 The edmx:DataServices Element 11](#_Toc320467249)

[3.2.1 The metadata:DataServiceVersion Attribute 11](#_Toc320467250)

[3.3 The edmx:Reference Element 11](#_Toc320467251)

[3.3.1 The edmx:Url Attribute 11](#_Toc320467252)

[3.4 The edmx:AnnotationsReference Element 12](#_Toc320467253)

[3.4.1 The edmx:Url Attribute 12](#_Toc320467254)

[3.5 The edmx:Include Element 12](#_Toc320467255)

[3.5.1 The edmx:TermNamespace Attribute 13](#_Toc320467256)

[3.5.2 The edmx:Qualifier Attribute 13](#_Toc320467257)

[4 Schema Constructs 13](#_Toc320467258)

[4.1 The edm:Schema Element 13](#_Toc320467259)

[4.1.1 The edm:Namespace Attribute 14](#_Toc320467260)

[4.1.2 The edm:Alias Attribute 14](#_Toc320467261)

[4.2 The edm:Using Element 14](#_Toc320467262)

[4.2.1 The edm:Namespace Attribute 15](#_Toc320467263)

[4.2.2 The edm:Alias Attribute 15](#_Toc320467264)

[5 Properties 15](#_Toc320467265)

[5.1 The edm:Property Element 15](#_Toc320467266)

[5.1.1 The edm:Name Attribute 16](#_Toc320467267)

[5.2 The edm:Type Attribute 16](#_Toc320467268)

[5.2.1 The edm:Nullable Facet 16](#_Toc320467269)

[5.2.2 The edm:MaxLength Facet 16](#_Toc320467270)

[5.2.3 The edm:FixedLength Facet 16](#_Toc320467271)

[5.2.4 The edm:Precision Attribute 17](#_Toc320467272)

[5.2.5 The edm:Scale Attribute 17](#_Toc320467273)

[5.2.6 The edm:Unicode Attribute 17](#_Toc320467274)

[5.2.7 The edm:Collation Attribute 17](#_Toc320467275)

[5.2.8 The edm:SRID Attribute 18](#_Toc320467276)

[5.2.9 The edm:Default Attribute 18](#_Toc320467277)

[5.2.10 The edm:ConcurrencyMode Attribute 18](#_Toc320467278)

[6 Entity Type Constructs 18](#_Toc320467279)

[6.1 The edm:EntityType Element 19](#_Toc320467280)

[6.1.1 The edm:Name Attribute 19](#_Toc320467281)

[6.1.2 The edm:BaseType Attribute 19](#_Toc320467282)

[6.1.3 The edm:Abstract Attribute 20](#_Toc320467283)

[6.1.4 The edm:OpenType Attribute 20](#_Toc320467284)

[6.2 The edm:Key Element 20](#_Toc320467285)

[6.3 The edm:PropertyRef Element 21](#_Toc320467286)

[6.4 The edm:NavigationProperty Element 21](#_Toc320467287)

[6.4.1 The edm:Name Attribute 21](#_Toc320467288)

[6.4.2 The edm:Relationship Attribute 22](#_Toc320467289)

[6.4.3 The edm:ToRole Attribute 22](#_Toc320467290)

[6.4.4 The edm:FromRole Attribute 22](#_Toc320467291)

[6.4.5 The edm:ContainsTarget Attribute 22](#_Toc320467292)

[7 Complex Type Constructs 23](#_Toc320467293)

[7.1 The edm:ComplexType Element 24](#_Toc320467294)

[8 Enumeration Type Constructs 24](#_Toc320467295)

[8.1 The edm:Enum Element 24](#_Toc320467296)

[8.1.1 The edm:Name Attribute 24](#_Toc320467297)

[8.1.2 The edm:UnderlyingType Attribute 25](#_Toc320467298)

[8.1.3 The edm:IsFlags Attribute 25](#_Toc320467299)

[8.2 The edm:Member Element 25](#_Toc320467300)

[8.2.1 The edm:Name Attribute 25](#_Toc320467301)

[8.2.2 The edm:Value Attribute 25](#_Toc320467302)

[9 Association Constructs 26](#_Toc320467303)

[9.1 The edm:Association Element 26](#_Toc320467304)

[9.1.1 The edm:Name Attribute 26](#_Toc320467305)

[9.2 The edm:End Element 26](#_Toc320467306)

[9.2.1 The edm:Type Attribute 26](#_Toc320467307)

[9.2.2 The edm:Role Attribute 26](#_Toc320467308)

[9.2.3 The edm:Multiplicity Attribute 26](#_Toc320467309)

[9.3 The edm:OnDelete Element 27](#_Toc320467310)

[9.4 The edm:ReferentialConstraint Element 27](#_Toc320467311)

[9.5 The edm:Principal Element 27](#_Toc320467312)

[9.6 The edm:Dependent Element 28](#_Toc320467313)

[9.7 The edm:PropertyRef Element 28](#_Toc320467314)

[10 Entity Container Constructs 28](#_Toc320467315)

[10.1 The edm:EntityContainer Element 30](#_Toc320467316)

[10.2 The edm:EntitySet Element 30](#_Toc320467317)

[10.3 The edm:AssociationSet Element 30](#_Toc320467318)

[10.3.1 The edm:End Element 31](#_Toc320467319)

[10.4 The edm:FunctionImport Element 31](#_Toc320467320)

[10.4.1 The edm:Name Attribute 31](#_Toc320467321)

[10.4.2 The edm:ReturnType Attribute 31](#_Toc320467322)

[10.4.3 The edm:EntitySet Attribute 31](#_Toc320467323)

[10.4.4 The edm:EntitySetPath Attribute 32](#_Toc320467324)

[10.5 The edm:ReturnType Element 32](#_Toc320467325)

[10.6 The edm:Parameter Element 32](#_Toc320467326)

[10.6.1 The edm:Name Attribute 32](#_Toc320467327)

[10.6.2 The edm:Type Attribute 32](#_Toc320467328)

[10.6.3 The edm:Mode Attribute 32](#_Toc320467329)

[10.6.4 Parameter facets 33](#_Toc320467330)

[11 Vocabulary Concepts 33](#_Toc320467331)

[12 Vocabulary Terms 34](#_Toc320467332)

[12.1 edm:EntityType and edm:ComplexType Terms 34](#_Toc320467333)

[12.2 The edm:ValueTerm Element 34](#_Toc320467334)

[13 Vocabulary Annotations 35](#_Toc320467335)

[13.1 The edm:TypeAnnotation Element 35](#_Toc320467336)

[13.2 The edm:PropertyValue Element 35](#_Toc320467337)

[13.3 The edm:ValueAnnotation Element 36](#_Toc320467338)

[13.4 The edm:Qualifier Attribute 36](#_Toc320467339)

[14 Vocabulary Expressions 36](#_Toc320467340)

[14.1 Constant Expressions 37](#_Toc320467341)

[14.1.1 The edm:Binary Constant Expression 37](#_Toc320467342)

[14.1.2 The edm:Bool Constant Expression 37](#_Toc320467343)

[14.1.3 The edm:DateTime Constant Expression 37](#_Toc320467344)

[14.1.4 The edm:DateTimeOffset Constant Expression 38](#_Toc320467345)

[14.1.5 The edm:Decimal Constant Expression 38](#_Toc320467346)

[14.1.6 The edm:Float Constant Expression 38](#_Toc320467347)

[14.1.7 The edm:Guid Constant Expression 39](#_Toc320467348)

[14.1.8 The edm:Int Constant Expression 39](#_Toc320467349)

[14.1.9 The edm:String Constant Expression 39](#_Toc320467350)

[14.1.10 The edm:Time Constant Expression 40](#_Toc320467351)

[14.2 Dynamic Expressions 40](#_Toc320467352)

[14.2.1 The edm:Apply Expression 40](#_Toc320467353)

[14.2.2 The edm:AssertType Expression 40](#_Toc320467354)

[14.2.3 The edm:Collection Expression 41](#_Toc320467355)

[14.2.4 The edm:EntitySetReference Expression 41](#_Toc320467356)

[14.2.5 The edm:EnumMemberReference Expression 41](#_Toc320467357)

[14.2.6 The edm:FunctionReference Expression 42](#_Toc320467358)

[14.2.7 The edm:If Expression 42](#_Toc320467359)

[14.2.8 The edm:IsType Expression 43](#_Toc320467360)

[14.2.9 The edm:LabeledElement Expression 43](#_Toc320467361)

[14.2.10 The edm:LabeledElementReference Expression 44](#_Toc320467362)

[14.2.11 The edm:Null Expression 44](#_Toc320467363)

[14.2.12 The edm:ParameterReference Expression 44](#_Toc320467364)

[14.2.13 The edm:Path Expression 44](#_Toc320467365)

[14.2.14 The edm:PropertyReference Expression 45](#_Toc320467366)

[14.2.15 The edm:Record Expression 45](#_Toc320467367)

[14.2.16 The edm:ValueTermReference Expression 45](#_Toc320467368)

[15 CSDL Examples 45](#_Toc320467369)

[15.1 Title for example 45](#_Toc320467370)

[16 ABNF for CSDL 47](#_Toc320467371)

# Appendix A: Formal Common Schema Definition Language (CSDL)

OData services are described by an Entity Data Model (EDM). Common Schema Definition Language (CSDL) defines an XML-based representation of the entity model exposed by an OData service. CSDL is based on standards defined in <ref>XML 1.0</ref> and <ref>XMLSCHEMA</ref>.

An OData service SHOULD provide a CSDL description of its entity model when a client requests a description of the entity model by sending a GET request to <serviceRoot>/$metadata. $metadata MUST wrap the CSDL document in an Edmx wrapper.

# Common Schema Definition Language (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.

## Entity Data Model for Data Services Packaging (EDMX) Namespace

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://schemas.microsoft.com/ado/2007/06/edmx.

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.

## Entity Data Model (EDM) Namespace

Elements and attributes that define the entity model exposed by the OData Service are qualified with the Entity Data Model namespace: http://schemas.microsoft.com/ado/2009/11/edm.

Prior versions of CSDL used the following namespaces for EDM:

* http://schemas.microsoft.com/ado/2006/04/edm
* http://schemas.microsoft.com/ado/2007/05/edm
* http://schemas.microsoft.com/ado/2008/01/edm
* http://schemas.microsoft.com/ado/2008/09/edm
* http://schemas.microsoft.com/ado/2009/11/edm

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

## Data Service Metadata Namespace

Elements and attributes specific to how the entity model is exposed as an OData Service are qualified with the Data Service Metadata namespace: http://schemas.microsoft.com/ado/2007/08/DataServices/Metadata.

In this specification the namespace prefix "metadata" is used to represent the Data Service Metadata namespace, however the prefix name is not prescriptive.

# Common Characteristics of Entity Model Elements

A typical entity model for an OData service contains one or more model elements. Some of these elements share a few common characteristics.

## Nominal Elements

Model elements can be nominal in nature. A nominal element has a name of the type <simpleIdentifier> that in combination with a <namespace> produces a fully qualified name of the form <namespaceQualifiedIdentifier>. The <namespaceQualifiedIdentifier> MUST be unique as it facilitates references to the element from other parts of the model.

When referring to nominal elements, the reference can use either of the following:

* Fully qualified name
* Alias qualified name

If the nominal element is unambiguous, the reference can simply use the name of the element.

Consider the following example:

<Schema  
 xmlns=http://schemas.microsoft.com/ado/2006/04/edm  
 xmlns:m=http://schemas.microsoft.com/ado/2007/08/dataservices/metadata  
 xmlns:d=http://schemas.microsoft.com/ado/2007/08/dataservices  
 Namespace="org.odata" Alias="odata">  
 <ComplexType Name="Address">  
 </ComplexType>  
</Schema>

The various ways of referring to the nominal element are:

* References in any namespace can use the fully qualified name, for example, org.odata.Address
* References in any namespace can specify an alias and use an alias qualified name, for example, odata.Address
* References in org.odata can simply use the name, for example, Address

## Structured Elements

Structured elements are composed of other model elements. Structured elements are common in entity models as they are the typical means of representing entities in the OData service. edm:EntityType and edm:ComplexType are both structured elements. edm:RowType is less common but is also a structured element.

A structural property is an edm:Property that has one of the following types:

* Primitive
* edm:ComplexType
* edm:EnumType

## The edm:Documentation Element

The edm:Documentation element allows service authors to provide documentation for most model elements.

A model element MUST NOT contain more than one documentation element.

Refer to the <ref>XML schema</ref> for details on which model elements support documentation.

A documentation element MUST contain zero or one edm:Summary and zero or one edm:LongDescription elements. The summary and long description elements must contain text that serves as the summary or long description. If both a summary and long description are provided, the summary MUST precede the long description.

For example:

<EntityType Name="Product">   
 <Documentation>  
 <Summary>Product names, suppliers, prices, and units in stock.</Summary>  
 <LongDescription>...</LongDescription>  
 </Documentation>  
 ...  
</EntityType>

## Vocabulary Annotations

Many model elements can be annotated with additional information with the edm:TypeAnnotation and edm:ValueAnnotation elements.

A model element MUST NOT specify more than one type annotation or value annotation for a given type term or value term.

Vocabulary annotations may 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 the <ref>XML schema</ref> for details on which model elements support vocabulary annotations.

## Custom Annotations

CSDL allows custom annotations to be attached to many model elements. This allows CSDL to be extended with markup to help various runtimes.

For instance, the following custom annotation attributes indicate that the entity container supports lazy loading:

<EntityContainer Name="ModelContainer" annotation:LazyLoadingEnabled="true">  
 ...  
</EntityContainer>

In the following example, the entity type has two access control entries specified with custom annotation elements:

<EntityType Name="Content">  
 ...  
 <RS:Security>  
 <RS:ACE Principal="S-0-123-1321" Rights="+R+W"/>  
 <RS:ACE Principal="S-0-123-2321" Rights="-R-W"/>  
 </RS:Security>  
</EntityType>

Custom annotations can appear in attribute form or element form. Custom annotations MUST NOT use a namespace that is on the list of reserved namespaces for CSDL. This includes the following namespaces:

* http://schemas.microsoft.com/ado/2007/06/edmx
* http://schemas.microsoft.com/ado/2007/08/dataservices/metadata
* http://schemas.microsoft.com/ado/2009/11/edm
* http://schemas.microsoft.com/ado/2008/09/edm
* http://schemas.microsoft.com/ado/2008/01/edm
* http://schemas.microsoft.com/ado/2007/05/edm
* http://schemas.microsoft.com/ado/2006/04/edm

Refer to the XML schema for details on which elements support custom annotations.

## Primitive Types

Model elements are composed of other model elements and primitive types. CSDL defines the following fully qualified primitive types:

* Edm.Binary
* Edm.Binary
* Edm.Boolean
* Edm.Byte
* Edm.DateTime
* Edm.Decimal
* Edm.Double
* Edm.Single
* Edm.Guid
* Edm.Int16
* Edm.Int32
* Edm.Int64
* Edm.SByte
* Edm.String
* Edm.Time
* Edm.DateTimeOffset
* Edm.Geography
* Edm.GeographyPoint
* Edm.GeographyLineString
* Edm.GeographyPolygon
* Edm.GeographyMultiPoint
* Edm.GeographyMultiLineString
* Edm.GeographyMultiPolygon
* Edm.GeographyCollection
* Edm.Geometry
* Edm.GeometryPoint
* Edm.GeometryLineString
* Edm.GeometryPolygon
* Edm.GeometryMultiPoint
* Edm.GeometryMultiLineString
* Edm.GeometryMultiPolygon
* Edm.GeometryCollection

# Entity Model Wrapper Constructs

An Entity Model Wrapper serves as the aggregation root for the schemas that describe the entity model exposed by the OData Service.

## The edmx:Edmx Element

An OData service exposes a single entity model. A CSDL description of the entity model can be requested from $metadata.

The document returned by $metadata MUST contain a single root edmx:Edmx element. This element MUST contain a single direct child edmx:DataServices element. edmx:DataServices describes the entity model(s) exposed by the OData service.

In addition, edmx:Edmx may have zero or more edmx:Reference elements and zero or more edmx:AnnotationsReference elements. edmx:Reference elements specify the location of schemas referenced by the OData service. edmx:AnnotationsReference elements specify the location of annotations to be applied to the OData service.

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

<edmx:Edmx xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx" Version="1.0">  
 <edmx:DataServices xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" m:DataServiceVersion="2.0">

<Schema ... />  
 </edmx:DataServices>  
</edmx:Edmx>

### The Version Attribute

The Version attribute MUST be present on the edmx:Edmx element. See <xmlschema> for details.

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 "1.0" of the EDMX Wrapper.

## The edmx:DataServices Element

The edmx:DataServices element contains zero or more edm:Schema elements which define the schema(s) exposed by the OData service.

### The metadata:DataServiceVersion Attribute

The metadata:DataServiceVersion attribute describes the version of OData protocol required to consume the service. This version of the specification defines the following valid data service version values: “1.0”, “2.0”, and “3.0”, corresponding to OData protocol versions 1.0, 2.0 and 3.0 respectively.

## The edmx:Reference Element

The edmx:Reference element specifies external entity models referenced by this EDMX. Referenced models are available in their entirety to referencing models. All entity types, complex types and other named elements in a referenced model can be accessed from a referencing model.

The following example demonstrates usage of edmx:Reference to reference entity models that contain entity types and complex types that are used as vocabulary terms:

<?xml version="1.0" encoding="UTF-8" standalone="true"?>  
<edmx:Edmx xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx" Version="1.0">  
 <edmx:Reference Url="http://vocabs.odata.org/capabilities/v1.0" />  
 <edmx:Reference Url="http://vocabs.odata.org/display/v1.0" />  
 <edmx:DataServices ...>  
</edmx:Edmx>

### The edmx:Url Attribute

The edmx:Reference element MUST specify an edmx:Url attribute. The edmx:Url attribute uniquely identifies a model. The URL may be backed by a CSDL describing the referenced model. Alternatively, the URL may be used to load a well-known model from a different location.

## The edmx:AnnotationsReference Element

The edmx:AnnotationsReference element specifies the location of an external document that contains annotations for this entity model. Only edm:Annotation, edm:TypeAnnotation and edm:ValueAnnotation elements will be read from the referenced model.

The edmx:AnnotationsReference element MUST contain one or more edmx:Include elements that specify the annotations to include from the target document.

The following example demonstrates usage of edmx:AnnotationsReference to reference documents that contain annotations:

<?xml version="1.0" encoding="UTF-8" standalone="true"?>  
<edmx:Edmx xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx" Version="1.0">  
 <edmx:AnnotationsReference Url="http://odata.org/ann/a">  
 <edmx:Include />  
 </edmx:AnnotationsReference>  
 <edmx:AnnotationsReference Url="http://odata.org/ann/b">  
 <edmx:Include TermNamespace="org.odata.validation" />  
 <edmx:Include TermNamespace="org.odata.display" Qualifier="Slate" />  
 </edmx:AnnotationsReference>  
 <edmx:DataServices ...>  
</edmx:Edmx>

All annotations from http://odata.org/ann/a are included. For http://odata.org/ann/b, only the following annotations are included:

* Annotations that use a term from org.odata.validation
* Annotations that use a term from org.odata.display and specify a Slate qualifier

### The edmx:Url Attribute

The edmx:AnnotationsReference element MUST specify an edmx:Url attribute. The value of the edmx:Url attribute uniquely identifies a model. The URL may be backed by a CSDL describing the referenced model. Alternatively, the URL may be used to load a well-known model from a different location.

## The edmx:Include Element

The edmx:Include element specifies which annotations to include from an edmx:AnnotationsReference. An edmx:Include that does not have an edmx:TermNamespace attribute or an edmx:Qualifier attribute includes all annotations within the document. If both TermNamespace and Qualifier have values, only annotations that meet both restrictions will be included.

### The edmx:TermNamespace Attribute

An edmx:Include element MAY have a value for the edmx: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 edmx:Include element will add the set of annotations that apply terms from the namespace in the value. The edmx:TermNamespace attribute also provides consumers insight about what namespaces are used in the annotations document. If there are no edmx:Include elements that have a term namespace of interest to the consumer, the consumer can opt to not download the document.

### The edmx:Qualifier Attribute

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

If a value is supplied, the edmx:Include element will add the set of annotations that apply the qualifier in the value. The edmx:Qualifier attribute also provides consumers insight about which qualifiers are used in the annotations document. If there are no edmx:Include elements that have a qualifier of interest to the consumer, the consumer can opt to not download the document.

# Schema Constructs

Each entity model exposed by the OData service is described one or more schemas. The schema acts as a container for all of the entity types, complex types and other model elements that make up an entity model.

## The edm:Schema Element

The Schema is the root of an entity model exposed by an OData service. Although an edmx:DataServices element contains zero or more Schema elements, many OData services will contain exactly one schema.

A Schema element contains zero or more of the following elements:

* edm:Annotations
* edm:Association
* edm:ComplexType
* edm:EntityContainer
* edm:EntityType
* edm:EnumType
* edm:Function
* edm:Using
* edm:ValueTerm

### The edm:Namespace Attribute

A schema is identified by the value of the Namespace attribute. The schema’s namespace is combined with the name of elements in the entity model to create unique names.

Identifiers that are used to name types MUST be unique within a namespace to prevent ambiguity. See <ref>Nominal Elements</ref> for more detail.

A schema that contains model elements other than vocabulary annotations MUST specify a <namespace> value for the Namespace attribute. A schema that contains only vocabulary annotations MAY specify a <namespace> value for the Namespace attribute.

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

### The edm:Alias Attribute

A schema MAY provide an <simpleIdentifier> value for the Alias attribute. An alias allows a CSDL document to qualify nominal elements with a short string rather than a long namespace. For instance, org.odata.vocabularies.display may simply have an alias of Self. An alias qualified name is resolved to a fully qualified name by examining aliases on edm:Using and Schema elements.

An alias is scoped to the container in which it is declared. For example, a model referencing an annotations document cannot use any aliases defined in that annotations document. A referencing model defines its own aliases with the Using element.

## The edm:Using Element

The Using element imports the contents of a specified namespace. A using element binds an alias to the namespace of any entity model.

Importing the contents of another model with a using element may alter the importing model. For instance, a model may import an entity model containing a type derived from a type in the importing model. In that case an edm:EntitySet in the importing model may return either type.

### The edm:Namespace Attribute

A using element MUST provide a <namespace> value to the Namespace attribute. The value provided to the Namespace attribute SHOULD match the namespace of an entity model that is in scope.

### The edm:Alias Attribute

A using element MUST define a <simpleIdentifier> value for the Alias attribute. An alias allows a CSDL model to substitute a short string for a long namespace. For instance, org.odata.vocabularies.display may be bound to an alias of display. An alias qualified name is resolved to a fully qualified name by examining aliases on edm:Using and edm:Schema elements.

# Properties

As mentioned in <ref>Structured Elements</ref>, structured elements are composed of other model elements. Structured elements expose a collection of zero or more edm:Property 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 null.

## The edm:Property Element

A Property element allows the construction of structured elements from a scalar value or a collection of scalar values.

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.

### The edm:Name Attribute

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 value of the name attribute MUST be unique within the set of properties and navigation properties for the type and any of its base types.

## The edm:Type Attribute

An edm: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 of the form <anyKeylessTypeReference>. The value of the type attribute MUST resolve to a complex type or a primitive type, or a collection of complex or primitive types.

## Property Facets

Facets apply to the nominal type referenced in the element where the facet is declared. In the following example, the Nullable facet applies to the DateTime type.

<Property Name="SuggestedTimes" Type="Collection(DateTime)" Nullable="true" />

In the following example the Nullable facet MUST be placed on the child element that references the DateTime type. Facets MUST NOT be applied to Collection type references.

<ReturnType>  
 <Collection>  
 <TypeRef Type="DateTime" Nullable="true" />  
 </Collection>  
</ReturnType>

### The edm:Nullable Facet

Any edm:Property MAY define a <boolean> value for the Nullable facet. The value of this facet 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.

### The edm:MaxLength Facet

A binary, stream or string edm:Property MAY define a <nonNegativeIntegral> value for the MaxLength facet. The value of this facet specifies the maximum length of the value of the property on a type instance.

### The edm:FixedLength Facet

A binary, stream or string edm:Property MAY define a <nonNegativeIntegral> value for the FixedLength facet. The value of this facet specifies the size of the array used to store the value of the property on a type instance.

### The edm:Precision Attribute

A temporal or decimal edm:Property MAY define a <nonNegativeIntegral> 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. 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.

### The edm:Scale Attribute

A decimal edm:Property MAY define a <nonNegativeIntegral> value for the Scale attribute. The value of this attribute specifies the maximum number of digits allowed to the right of the decimal point in the value of the property on a type instance.

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

### The edm:Unicode Attribute

A string edm: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 defined for this attribute, the value defaults to true.

### The edm:Collation Attribute

A string edm:Property MAY define a value for the Collation attribute. The value of this attribute specifies a collation sequence that can be used for comparison and ordering operations.

The value of the collation attribute MUST be one of the following:

* Binary
* Boolean
* Byte
* DateTime
* DateTimeOffset
* Time
* Decimal
* Double
* Single
* Guid
* Int16
* Int32
* Int64
* String
* SByte

### The edm:SRID Attribute

A spatial 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 <nonNegativeInt32> 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) [[EPSG](http://go.microsoft.com/fwlink/?linkid=148018)].

### The edm:DefaultValue Facet

A string property MAY define a value for the DefaultValue attribute. The value of this attribute determines the value of the property on new type instances.

### The edm:ConcurrencyMode Attribute

An edm:Property MAY define a value for the ConcurrencyMode attribute. The value of this attribute indicates how concurrency should be handled for the property.

The value of the concurrency mode attribute MUST be None or Fixed. If no value is specified, the value defaults to None.

When used on a property of an entity type, the concurrency mode attribute specifies that the value of that property SHOULD be used for optimistic concurrency checks.

The concurrency mode attribute MUST NOT be applied to any properties of a complex type.

The concurrency mode attribute MUST NOT be applied to properties whose type is a complex type.

# Entity Type Constructs

Entity types are nominal structured model elements with a key that consists of one or more references to structural properties. An entity type by definition has an independent existence and can be created, updated or deleted independently of any other types. 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 or complex type, or a collection of primitive or complex types. A navigation property is a named reference to another entity type or collection of entity types. All properties MUST have a unique name. 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" Relationship="ODataDemo.Product\_Category\_Category\_Products" FromRole="Product\_Category" ToRole="Category\_Products"/>  
 <NavigationProperty Name="Supplier" Relationship="ODataDemo.Product\_Supplier\_Supplier\_Products" FromRole="Product\_Supplier" ToRole="Supplier\_Products"/>  
</EntityType>

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

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

## The edm:EntityType Element

The EntityType element represents an entity type in the entity model.

An entity type MUST contain exactly one edm:Key element or specify a value for the edm:BaseType attribute, but not both.

If no base type is specified, the edm:EntityType element MUST contain one or more edm:Property elements describing the properties of the entity type. The edm:EntityType element also can contain zero or more edm:NavigationProperty elements.

### The edm:Name Attribute

A value of the form <simpleIdentifier> MUST be provided for the Name attribute because an entity type is a nominal element. The value identifies the entity type and MUST be unique within the entity type’s namespace.

### The edm:BaseType Attribute

An entity type can inherit from another entity type by specifying a <singleEntityTypeReference> value for the BaseType attribute.

An entity type that provides a value for the base type attribute MUST NOT declare a key with the edm:Key element.

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.

### The edm:Abstract Attribute

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.

### The edm:OpenType Attribute

An entity type MAY indicate that it can be freely extended by providing a <boolean> value of true to the OpenType attribute. An open entity type allows entity instances to add properties dynamically simply by adding uniquely named values to the payload.

If no value is provided for the open type attribute, the value of the open type attribute is set to false.

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

## The edm:Key Element

An entity type must be uniquely identifiable. If an entity type does not specify a base type, the entity type MUST contain exactly one Key element. An entity type’s key refers to the set of properties that uniquely identify an instance of the entity type.

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 primitives.

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 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>

## The edm:PropertyRef Element

The PropertyRef element provides an edm:Key with a reference to a single property of an entity type. A <simpleIdentifier> value MUST be supplied to the Name attribute. This value MUST resolve to one of the properties of the entity type.

## The edm:NavigationProperty Element

A navigation property allows navigation from an entity to one or more 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" ToRole="Category" FromRole="Product" Relationship="ODataDemo.ProductCategory"/>  
 <NavigationProperty Name="Supplier" ToRole="Supplier" FromRole="Product" Relationship="ODataDemo.ProductSupplier"/>  
</EntityType>  
<EntityType Name="Category">  
 ...  
 <NavigationProperty Name="Products" ToRole="Product" FromRole="Category" Relationship="ODataDemo.ProductCategory"/>  
</EntityType>

### The edm:Name Attribute

The navigation property MUST provide a <simpleIdentifier> value to the Name attribute. The name attribute is a meaningful string that characterizes the relationship when navigating from the entity that declared the navigation property to the related entity.

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

### The edm:Relationship Attribute

The Relationship attribute MUST be given <simpleIdentifier> or <qualifiedIdentifier>. The value of the attribute MUST be the same as the name of an edm:Association in the entity model.

### The edm:ToRole Attribute

The navigation property MUST provide a <simpleIdentifier> value to the ToRole attribute. The ToRole attribute is a name used to refer to the destination of the navigation property.

The value provided to the ToRole attribute MUST be the same as one of the edm:Role names on the corresponding edm:Association.

### The edm:FromRole Attribute

The navigation property MUST provide a <simpleIdentifier> value to the FromRole attribute. The FromRole attribute is a name used to refer to the origin of the navigation property.

The value provided to the FromRole attribute MUST be the same as one of the edm:Role names on the corresponding edm:Association.

### The edm:ContainsTarget Attribute

A <boolean> value MAY be assigned 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, the entity type to which the navigation property belongs is said to contain the destination of the navigation property.

TODO: WORDSMITH

“The [EntityType](#z6875ce6c837c4cea8e35441dc2366008) that declares the NavigationProperty is the container EntityType.

The AssociationType that is specified in the containment NavigationProperty is the containment AssociationType.

The EntityType that is specified on the End element of the containment AssociationType, with the Name that is specified by the containment NavigationProperty element's ToRole attribute, is the contained EntityType.

When the instances of both the contained entity and the container entity reside in the same [EntitySet](#z4a09a48c1da34d8487b42b6c46731470), it is called recursive containment.

It MUST NOT be possible for an EntityType to contain itself by following more than one containment NavigationProperty.

The contained EntityType can have a NavigationProperty that navigates to the container EntityType via the containment AssociationType.

The End of the containment AssociationType that is specified by the ToRole attribute of the containment NavigationProperty can have any multiplicity.

For nonrecursive containment, the End of the containment AssociationType that is specified by the FromRole attribute of the containment NavigationProperty MUST have a multiplicity of '1'.

For recursive containment, the End of the containment AssociationType that is specified by the FromRole attribute of the containment NavigationProperty MUST have a multiplicity of '0..1'. The End that is specified by the ToRole MUST not have a multiplicity of '1' because this would lead to endless recursion.

An [AssociationSet](#z84fdfd027b124aa3a2eb51bab109f439)MUST have the same EntitySet on both ends if it is for a containment AssociationType that has either the same EntityType on both ends or an EntityType on one end that derives from the EntityType on the other end.

An EntitySet MUST NOT be bound by AssociationSet to more than one AssociationType via a containment NavigationProperty that indicates that the EntityType (or derived EntityTypes) of that EntitySet is contained.

Note  Because the EntityType of an EntitySet on an [AssociationSet End](#z3c3578f79de94e7b9a852ed690bab9e7) MUST be the same as or derived from the EntityTypes on the corresponding AssociationType End, the EntitySet MUST be either completely contained or completely noncontained.

”

# Complex Type Constructs

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

## The edm:ComplexType Element

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

The complex type MUST declare a <simpleIdentifier> value for the Name attribute as well as one or more edm:Property elements. Complex types MUST NOT have any navigation properties.

# Enumeration Type Constructs

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>

## The edm:EnumType Element

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

The enumeration type MUST provide a <simpleIdentifier> as the value of the Name attribute.

The edm:EnumType element contains zero or more child edm:Member elements enumerating the members of the enum.

### The edm:UnderlyingType Attribute

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

The enumeration type has an underlying type MUST be an <integralPrimitive>. If the underlying type is not specified, a 32-bit integer MUST be used as the underlying type.

### The edm:IsFlags Attribute

An enumeration type MAY specify a <boolean> value for the edm:IsFlags attribute. A value of <true> indicates that the enumeration type allows multiple members to be selected simultaneously.

## The edm:Member Element

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

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

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

### The edm:Name Attribute

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

### The edm:Value Attribute

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

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>

# Other Type Constructs

## Collection Types

A collection type is an alternative way to represent a collection of scalar, complex or entity types.

A collection type can be represented with attribute notation or element notation, as shown in these two equivalent examples:

<ReturnType Type="Collection(Edm.String) " />  
<ReturnType>  
 <CollectionType>  
 <TypeRef Type="Edm.String" />  
 </CollectionType>  
</ReturnType>

If specified with attribute notation, the collection type MUST be a collection of entity types, complex types or scalar types. If specified with element notation, the collection type can also be a collection of other collection types, reference types or row types.

### The edm:CollectionType Element

The CollectionType element represents a collection of other types in an entity model.

The CollectionType element can identify the types contained in the collection by specifying a <simpleIdentifier> or <qualifiedIdentifier> value for the ElementType attribute.

Alternatively the collection type can identify the types contained in the collection by specifying one of the following child elements:

* edm:CollectionType
* edm:ReferenceType
* edm:RowType
* edm:TypeRef

The collection type MUST identify the types contained in the collection with exactly one of the methods indicated above.

The collection type can define relevant facets for scalar types.

## The edm:TypeRef Element

The TypeRef element is used to reference a nominal type.

The TypeRef element MUST provide a <simpleIdentifier> or <qualifiedIdentifier> value for the Type attribute.

The type ref can define relevant facets for scalar types.

## Reference Types

A reference type specifies a reference to an entity instance. A reference to an entity instance is useful when a large number of entities would otherwise be returned.

A reference type can be specified with attribute notation or element notation, as shown in the following two examples:

<ReturnType Type="Ref(Self.Customer) " />  
<ReturnType>  
 <CollectionType>  
 <ReferenceType Type="Self.Customer" />  
 </CollectionType>  
</ReturnType>

### The edm:ReferenceType Element

The ReferenceType element represents a reference type in an entity model.

A reference type MUST specify a <singleEntityTypeReference> value for the Type attribute. The value of this attribute names the type for which the reference type contains key information.

## Row Types

A row type is the only structural type that is not nominal in nature. Because it does not have a name, the row type MUST be used inline. Row types are frequently used as the structure of a function import’s return type.

### The edm:RowType Element

The RowType element represents a structural type without a name. The row type MUST declare one or more edm:Property elements that define its structure.

# Association Constructs

Associations provide the fundamental definition for a relationship between two entity types. An association MUST have a name and two ends, each with their own cardinality. In some cases, an association also has a referential constraint. A referential constraint asserts that the entity type playing the principal role in an association MUST exist for the entity type playing the dependent role to exist.

A simple association might look like this:

<Association Name="ProductCategory">   
 <End Type="ODataDemo.Product" Multiplicity="\*" Role="Product"/>  
 <End Type="ODataDemo.Category" Multiplicity="0..1" Role="Category"/>  
</Association>

## The edm:Association Element

The Association element represents an association in an entity model. The association MUST provide a <simpleIdentifier> value for the Name attribute.

An association MUST contain exactly two edm:End elements that represent the respective ends of the association. The association MUST also specify zero or one edm:ReferentialConstraint elements.

## The edm:End Element

An association has exactly two ends, each of which plays a role, has its own multiplicity and refers to the entity type at that end of the association. The End element describes one of the ends of an association.

Each end MUST also have zero or one edm:OnDelete elements to specify operational behavior.

### The edm:Type Attribute

Each end of the association MUST specify the entity type attached to that end. The value of the Type attribute must be a <singleEntityTypeReference>. The value of the type attribute MUST resolve to an entity type in the entity model.

### The edm:Role Attribute

The Role attribute allows the association end to be bound to a navigation property. The association end MAY assign a <simpleIdentifier> value to the Role attribute.

### The edm:Multiplicity Attribute

The Multiplicity attribute defines the cardinality of the association end. The value of the attribute MUST be one of the following:

* 0..1 – zero or one
* 1 – exactly one
* \* – zero or more

## The edm:OnDelete Element

The OnDelete element prescribes the action that should be taken when the entity on the opposing end of the association is deleted.

The OnDelete element MUST define a value for the Action attribute. The value assigned to the action attribute MUST be Cascade or None.

## The edm:ReferentialConstraint Element

A referential constraint asserts that the entity on the principal end of the referential constraint must exist in order for the entity on the dependent end to exist. This assertion is established by the ReferentialConstraint element.

A referential constraint MUST contain exactly one edm:Principal element and exactly one edm:Dependent element.

In the example that follows, the category must exist for a product in that category to exist:

<Association Name="ProductCategory">   
 <End Type="ODataDemo.Product" Multiplicity="\*" Role="Product"/>  
 <End Type="ODataDemo.Category" Multiplicity="0..1" Role="Category"/>  
 <ReferentialConstraint>  
 <Principal Role="Category">  
 <PropertyRef Name="CategoryID" />  
 </Principal>  
 <Dependent Role="Product">  
 <PropertyRef Name="CategoryID" />  
 </Dependent>  
 </ReferentialConstraint>  
</Association>

## The edm:Principal Element

The Principal element represents the dominant end of the association. The entity on this end of the association may exist independently of the entity on the dependent end of the association.

The Principal element MUST provide a <simpleIdentifier> value for the Role attribute. This value MUST have the same value as the Role attribute of one of the association ends. The multiplicity of that association end MUST be 1 or 0..1.

The principal end of the referential constraint MUST contain one or more edm:PropertyRef elements that name the key properties of the principal entity type.

## The edm:Dependent Element

The Dependent element represents the subordinate end of the association. The entity on this end of the association may only exist if an entity on the opposite end of the association exists.

The edm:Dependent element MUST provide a <simpleIdentifier> value for the Role attribute. This value MUST have the same value as the Role attribute of one of the association ends.

## The edm:PropertyRef Element

The PropertyRef elements indicate the fields that take part in the referential constraint on each entity.

The edm:Principal element and the edm:Dependent element MUST each contain the same number of PropertyRef elements. The property references MUST each be assigned <simpleIdentifier> values and be ordered consistently in the principal and dependent elements. The property references MUST have the same data types in the principal and dependent elements.

The property references for the principal entity MUST be the same property references specified in the edm:Key of the principal entity type, however they need not share the same order.

The following example is NOT valid because the order of the edm:PropertyRef elements is inconsistent on the principal and dependent ends of the referential constraint:

<Association Name="OrderLineDetails">   
 <End Type="ODataDemo.OrderLineDetail" Multiplicity="\*" Role="Detail"/>  
 <End Type="ODataDemo.OrderLine" Multiplicity="0..1" Role="OrderLine"/>  
 <ReferentialConstraint>  
 <Principal Role="OrderLine">  
 <PropertyRef Name="OrderID" />  
 <PropertyRef Name="LineNumber" />  
 </Principal>  
 <Dependent Role="Detail">  
 <PropertyRef Name="LineNumber" />  
 <PropertyRef Name="OrderID" />  
 </ Dependent>  
 </ReferentialConstraint>  
</Association>

# Entity Container Constructs

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, entity sets and association sets.

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

<EntitySet Name="Products" EntityType="ODataDemo.Product"/>  
<EntitySet Name="Categories" EntityType="ODataDemo.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="ODataDemo.Product"/>  
<EntitySet Name="DiscontinuedProducts" EntityType="ODataDemo.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.

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

An association set is a nominal element that disambiguates the entity set used by an association end. In the preceding example an ODataDemo.Product could be exposed through the Products entity set or the DiscontinuedProducts entity set. An association set is required to clarify which entity set is used for any association end that refers to ODataDemo.Product.

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" EntitySet="Products" ReturnType="Collection(ODataDemo.Product)">  
 <Parameter Name="fiscalYear" Mode="In" Type="String" />  
</FunctionImport>

An entity container is the entity model equivalent of a single data store. An entity container aggregates entity sets, association sets and function imports.

A full example of an entity container is as follows:

<EntityContainer Name="DemoService">  
 <EntitySet Name="Products" EntityType="ODataDemo.Product"/>  
 <EntitySet Name="Categories" EntityType="ODataDemo.Category"/>  
 <EntitySet Name="Suppliers" EntityType="ODataDemo.Supplier"/>  
 <AssociationSet Name="ProductCategorySet" Association="ODataDemo.ProductCategory">  
 <End Role="Product" EntitySet="Products"/>  
 <End Role="Category" EntitySet="Categories"/>  
 </AssociationSet>  
 <AssociationSet Name="ProductSupplierSet" Association="ODataDemo.ProductSupplier">  
 <End Role="Product" EntitySet="Products"/>  
 <End Role="Supplier" EntitySet="Suppliers"/>  
 </AssociationSet>  
 <FunctionImport Name="GetProductsByRating" EntitySet="Products" ReturnType="Collection(ODataDemo.Product)">  
 <Parameter Name="rating" Type="Edm.Int32" Mode="In"/>  
 </FunctionImport>  
</EntityContainer>

## The edm:EntityContainer Element

The 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:AssociationSet or edm:FunctionImport elements.

The entity container MUST provide a unique <simpleIdentifier> value for the Name attribute.

An entity container MAY provide a <qualifiedIdentifier> value for the Extends attribute. The value provided to the extends attribute MUST resolve to an entity container in the entity model. All of the children in the extending entity container are added to the children of the extended entity container.

## The edm:EntitySet Element

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

An entity set MUST provide a <simpleIdentifier> value for the Name attribute. An entity set also has a EntityType attribute that MUST be provided with a <singleEntityTypeReference> that resolves to an entity type in the model. Each entity type in the model may have zero or more entity sets that reference the entity type.

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

## The edm:AssociationSet Element

The AssociationSet element is a nominal element that represents an association set in an entity model.

An association set MUST provide a <simpleIdentifier> value for the Name attribute. An association set also has an Association attribute that MUST be provided with a <qualifiedIdentifier> that resolves to an association in the entity model.

An association set can contain zero, one or two edm:End elements.

### The edm:End Element

An End element MUST be specified if the corresponding end of the association refers to an entity type that is exposed in more than one entity set.

An association set end MUST provide a <simpleIdentifier> value for the Role attribute that is the same as the value of one of the association ends’ Role attribute.

The End element MUST also provide a <simpleIdentifier> or <qualifiedIdentifier> value for the EntitySet attribute. The entity set that is named MUST expose the entity type bound by the corresponding end of the association.

## The edm:FunctionImport Element

The edm:FunctionImport element is a nominal element that represents a store function in an entity model. The function may return a single or collection of primitive types or nominal structured types.

A function import MUST provide a <simpleIdentifier> value for the Name attribute.

The function import MUST specify zero or more return types. If the function import has only one return type, the return type can be specified using the edm:ReturnType attribute. If the function import has more than one return type, the return type MUST be specified using multiple edm:ReturnType elements. All return types must be scalar, entity or complex types or a collection of scalar, entity or complex types.

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

### The edm:ReturnType Attribute

If the return type is written with attribute notation, an <anyTypeIdentifier> value must be provided for the edm:ReturnType attribute.

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

### The edm:EntitySet Attribute

If the return type is an entity or a collection of entities, a <simpleIdentifier> value must be defined for the edm:EntitySet attribute that names the entity set to which the returned entities belong.

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

If the edm:EntitySet attribute is assigned a value, the edm:EntitySetPath MUST NOT be assigned a value.

### The edm:EntitySetPath Attribute

The function import MAY specify a value for the edm:EntitySetPath attribute if determination of the entity set for the return type is contingent on a parameter.

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

The first segment of the entity set path MUST have the same name as one of the function import’s parameters. The remaining segments of the entity set path MUST represent navigation or type casts.

A navigation segment simply names the <simpleIdentifier> of the navigation property to be traversed. A type cast segment names the <qualifiedIdentifier> of the entity type that should be returned from the type cast.

### The edm:IsSideEffecting Attribute

A function import can specify a <boolean> value for the IsSideEffecting attribute. If no value is specified for the IsSideEffecting attribute, the value defaults to true.

If the value of the IsSideEffecting attribute is true, the value of the edm:IsComposable attribute must be false.

### The edm:IsBindable Attribute

A function import can specify a <boolean> value for the IsBindable attribute. If no value is specified for the IsBindable attribute, the value defaults to false.

If the IsBindable attribute is set to true, the function import MUST contain at least one edm:Parameter element.

### The edm:IsComposable Attribute

A function import can specify a <boolean> value for the IsComposable attribute. If no value is specified for the IsComposable attribute, the value defaults to false.

If the value of the IsComposable attribute is true, the value of the edm:IsSideEffecting attribute must be false.

## The edm:ReturnType Element

If the return type is written with element notation, the function import MUST contain one or more ReturnType elements. The presence of multiple return type elements may occur when the store returns more than one result set for a given function.

If element notation is used, a similar set of attributes can be used to specify the return type of the function. The Type attribute corresponds to the edm:ReturnType attribute on edm:FunctionImport, and the names and functionality of the edm:EntitySet and edm:EntitySetPath attributes remain the same.

## The edm:Parameter Element

The 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 Parameter element.

The Name attribute MUST be used to assign a unique <simpleIdentifier> value to the parameter.

### The edm:Type Attribute

The parameter MUST indicate which set of types can be passed to the parameter by providing a <anyTypeReference> value for the Type attribute.

### The edm:Mode Attribute

A value of In, Out, or InOut MAY be provided to the Mode attribute. These values correspond to the modality of parameters passed to stored procedures in relational databases.

### Parameter Facets

A parameter may specify values for the edm:Nullable, edm:MaxLength, edm:Precision, edm:Scale, or edm:SRID attributes. The descriptions of these facets and their implications are covered elsewhere in this specification.

# Vocabulary Concepts

Vocabulary terms and annotations provide a means of semantically enriching an entity model and the type instances accessible through that entity model. Annotations can be used for two fundamental purposes:

* To extend type instances with additional information
* To enable new types to be instantiated from existing type instances

Type instances can be extended with additional information through the application of value annotations. A value annotation attaches a value term to a type instance and provides a means of calculating a value for the value term.

For example, the following entity type includes a value annotation that allows a display name to be calculated from an entity instance:

<EntityType Name="Category">  
 ...  
 <Property Name="Name" Nullable="true" Type="Edm.String"/>  
 <ValueAnnotation Term="display.DisplayName" Path="Name" />  
</EntityType>

Type annotations allow a different type to be instantiated from an instance of an entity type. The type annotation attaches a type term to a type instance and provides a means of instantiating the type named in the term.

For instance, the following entity type includes a type annotation that allows a SearchResult to be instantiated from a Product:

<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.DateTime"/>  
 <Property Name="Rating" Nullable="false" Type="Edm.Int32"/>  
 <Property Name="Price" Nullable="false" Type="Edm.Decimal"/>  
 ...  
 <TypeAnnotation Term="SearchVocabulary.SearchResult">  
 <PropertyValue Property="Title" Path="Name" />  
 <PropertyValue Property="Url">  
 <Apply Function="CommonFunctionsVocabulary.Concat">  
 <String>Products(</String>  
 <Path>ID</Path>  
 <String>)</String>  
 </Apply>  
 </PropertyValue>  
 <PropertyValue Property="Abstract">  
 <Path>Description</Path>  
 </PropertyValue>  
 </TypeAnnotation>  
</EntityType>

Type and value annotations can be defined at a universal or instance level. Universal annotations are applied directly in the entity model as described in CSDL. Instance annotations are applied to entity instances in the response payload.

Type and value annotations can be used to apply type and value terms respectively. These terms may be part of a well-known vocabulary or part of a company-specific or product-specific vocabulary.

# Vocabulary Terms

There are two types of vocabulary terms that can be applied: type terms and value terms.

A type term implies that a type can be instantiated from the annotated element. Either an edm:EntityType or an edm:ComplexType can be used as a type term. There are no special requirements for an edm:EntityType or an edm:ComplexType to be used as a type term.

A value term extends an existing instance with additional data. A value term is limited to a single field, however that field may be an entity type or a complex type.

## edm:EntityType and edm:ComplexType Terms

There are no special requirements for an edm:EntityType or an edm:ComplexType to be used as a type term. Type terms need not be in the same model as the type annotations or the annotated types.

## The edm:ValueTerm Element

The ValueTerm element represents a value term in an entity model.

The value term MUST provide a <simpleIdentifier> value for the Name attribute and a <simpleIdentifier> or <qualifiedIdentifier> value for the Type attribute. The name attribute allows the term to be applied with a value annotation. The type attribute indicates what type of value must be returned by the expression contained in the value annotation.

# Vocabulary Annotations

Vocabulary terms are used to annotate other model elements. The following model elements may be annotated with a vocabulary term:

* Annotations
* ComplexType
* EntitySet
* EntityType
* EnumType
* Function
* FunctionImport
* NavigationProperty
* Parameter
* Property
* ValueTerm

The most common usage of a type term is to annotate an entity type or complex type. In this situation, an instance of the type term may be created for each instance of the entity type or complex type the term is applied to.

Type terms can also be used to annotate a variety of other model elements including an entity set, a function import, or a navigation property.

In contrast to type annotations, which allow the instantiation of new types, a value annotation extends an existing type instance with additional information.

## The edm:Annotations Element

The Annotations element is used to apply a group of type or value annotations to a single model element.

An annotations element MUST assign a <simpleIdentifier> or <qualifiedIdentifier> value to the Target attribute. The value of the target attribute SHOULD resolve to a model element in the entity model.

An annotations element MUST contain zero or more edm:TypeAnnotation and edm:ValueAnnotation elements.

## The edm:TypeAnnotation Element

The TypeAnnotation element represents a type annotation. A type annotation binds a type term to a model element.

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

A type annotation MUST provide a <simpleIdentifier> or <qualifiedIdentifier> value for the Term attribute. The value of the term attribute SHOULD resolve to an entity type or complex type.

A type annotation MUST contain zero or more edm:PropertyValue elements.

## The edm:PropertyValue Element

The PropertyValue element supplies a value to a property on the type instantiated by a type annotation. The value is obtained by evaluating an expression.

The property value element MUST assign a <simpleIdentifier> value to the Property attribute. The value of the property attribute SHOULD resolve to a property on the term referenced by the type annotation.

A property value MUST contain exactly one expression. The expression can be provided using element notation or attribute notation.

## The edm:ValueAnnotation Element

The ValueAnnotation element represents a value annotation. A value annotation attaches a named value to a model element.

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

A value annotation MUST provide a <simpleIdentifier> or <qualifiedIdentifier> value for the Term attribute. The value of the term attribute SHOULD resolve to a value term.

A value annotation MUST contain at most one expression. The expression can be provided using element notation or attribute notation.

## The edm:Qualifier Attribute

The qualifier attribute allows service authors a means of conditionally applying a type annotation. For instance, the following value annotation hints that it should only be applied to slate devices:

<ValueAnnotation Term="org.odata.display.DisplayName" Path="FirstName" Qualifier="org.odata.formfactor.Slate">

The value of the Qualifier attribute is an arbitrary string.

An edm:Annotations element MUST provide at most one value for the Qualifier attribute.

Type or value annotations MUST provide at most one value for the Qualifier attribute. Type or value annotations that are children of an edm:Annotations element MUST NOT provide a value for the Qualifier attribute.

# Vocabulary Expressions

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

All vocabulary expressions may be specified as an element, for example:

<ValueAnnotation Term="org.odata.display.DisplayName">  
 <String>Customers</String>  
</ValueAnnotation>

The constant expressions and the Path expression also support attribute notation:

<ValueAnnotation Term="org.odata.display.DisplayName" String="Customers" />

## Constant Expressions

Constant expressions allow the service author to supply an unchanging value to a value term or property of a type term.

The following examples show two value annotations intended as user interface hints:

<EntitySet Name="Products" EntityType="ODataDemo.Product">  
 <ValueAnnotation Term="org.odata.display.DisplayName" String="Product Catalog" />  
</EntitySet>  
<EntitySet Name="Suppliers" EntityType="ODataDemo.Supplier">  
 <ValueAnnotation Term="org.odata.display.DisplayName" String="Supplier Directory" />  
</EntitySet>

### The edm:Binary Constant Expression

The Binary constant expression evaluates to a primitive binary value. A binary expression MUST be assigned a value of the type <xsHexBinary>.

A binary expression may be written with either element notation or attribute notation, as shown in the following examples:

<ValueAnnotation Term="org.odata.display.Thumbnail" Binary="3f3c6d78206c" />  
<ValueAnnotation Term="org.odata.display.Thumbnail">  
 <Binary>3f3c6d78206c</Binary>  
</ValueAnnotation>

### The edm:Bool Constant Expression

The Bool constant expression evaluates to a primitive boolean value. A boolean expression MUST be assigned a value of the type <xsBoolean>.

A boolean expression may be written with either element notation or attribute notation, as shown in the following examples:

<ValueAnnotation Term="org.odata.display.ReadOnly" Bool="true" />  
<ValueAnnotation Term="org.odata.display.ReadOnly">  
 <Bool>true</Bool>  
</ValueAnnotation>

### The edm:DateTime Constant Expression

The DateTime constant expression evaluates to a primitive date/time value. A date/time expression MUST be assigned a value of the type <xsDateTime>. The UTC offset portion of this value MUST be discarded.

A date/time expression may be written with either element notation or attribute notation, as shown in the following examples:

<ValueAnnotation Term="org.odata.display.LastUpdated" DateTime="2000-01-01T16:00:00.000" />  
<ValueAnnotation Term="org.odata.display.LastUpdated">  
 <DateTime>2000-01-01T16:00:00.000</DateTime>  
</ValueAnnotation>

### The edm:DateTimeOffset Constant Expression

The DateTimeOffset constant expression evaluates to a primitive date/time value with a UTC offset. A DateTimeOffset expression MUST be assigned a value of the type <xsDateTime>.

A DateTimeOffset expression may be written with either element notation or attribute notation, as shown in the following examples:

<ValueAnnotation Term="org.odata.display.LastUpdated" DateTimeOffset="2000-01-01T16:00:00.000Z-09:00" />  
<ValueAnnotation Term="org.odata.display.LastUpdated">  
 <DateTime>2000-01-01T16:00:00.000Z-09:00</DateTime>  
</ValueAnnotation>

### The edm:Decimal Constant Expression

The Decimal constant expression evaluates to a primitive decimal value. A decimal expression MUST be assigned a value of the type <xsDecimal>.

A decimal expression may be written with either element notation or attribute notation, as shown in the following examples:

<ValueAnnotation Term="org.odata.display.Width" Decimal="3.14" />  
<ValueAnnotation Term="org.odata.display.Width">  
 <Decimal>3.14</Decimal>  
</ValueAnnotation>

### The edm:Float Constant Expression

The Float constant expression evaluates to a primitive floating point (or double) value. A floating point expression MUST be assigned a value of the type <xsDouble>.

A floating point expression may be written with either element notation or attribute notation, as shown in the following examples:

<ValueAnnotation Term="org.odata.display.Width" Float="3.14" />  
<ValueAnnotation Term="org.odata.display.Width">  
 <Float>3.14</Float>  
</ValueAnnotation>

### The edm:Guid Constant Expression

The Guid constant expression evaluates to a primitive 32-character string value. A guid expression MUST be assigned a value of the type <xsGuid>.

A guid expression may be written with either element notation or attribute notation, as shown in the following examples:

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

### The edm:Int Constant Expression

The Int constant expression evaluates to a primitive integral value. An integral expression MUST be assigned a value of the type <xsInteger>.

An integral expression may be written with either element notation or attribute notation, as shown in the following examples:

<ValueAnnotation Term="org.odata.display.Width" Int="42" />  
<ValueAnnotation Term="org.odata.display.Width">  
 <Int>42</Int>  
</ValueAnnotation>

### The edm:String Constant Expression

The String constant expression evaluates to a primitive string value. A string expression MUST be assigned a value of the type <xsString>.

A string expression may be written with either element notation or attribute notation, as shown in the following examples:

<ValueAnnotation Term="org.odata.display.DisplayName" String="Product Catalog" />  
<ValueAnnotation Term="org.odata.display.DisplayName">  
 <String>Product Catalog</String>  
</ValueAnnotation>

### The edm:Time Constant Expression

The Time constant expression evaluates to a primitive time value. On platforms that do not support a primitive time value, the Time constant expression evaluates to a primitive date/time value. A time expression MUST be assigned a value of the type <xsTime>.

A time expression may be written with either element notation or attribute notation, as shown in the following examples:

<ValueAnnotation Term="org.odata.display.EndTime" Time="21:00:00-08:00" />  
<ValueAnnotation Term="org.odata.display.EndTime">  
 <Time>21:00:00-08:00</Time>  
</ValueAnnotation>

## Dynamic Expressions

### The edm:Apply Expression

The Apply expression enables a value to be obtained by applying a client-side function. An apply expression MUST assign a string value to the Function attribute. The value of the function attribute SHOULD be a <qualifiedIdentifier>. The value of the function attribute is used to locate the client-side function that should be applied.

The Apply expression MUST contain zero or more expressions. The expressions contained within the apply expression are used as parameters to the function.

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

<ValueAnnotation Term="org.odata.display.DisplayName">  
 <Apply Function="org.odata.commonfunctions.StringConcat">  
 <String>Product</String>  
 <String> </String>  
 <String>Catalog</String>  
 </Apply>  
</ValueAnnotation>

### The edm:AssertType Expression

The 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 assign a value of the type <anySingleTypeIdentifier> to the Type attribute.

The AssertType expression MUST contain exactly one expression. The expression contained within the assert type expression is used as a parameter to the type assertion.

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

<ValueAnnotation Term="org.odata.display.DisplayName">  
 <AssertType Type="Edm.String">  
 <String>Product Catalog</String>  
 </AssertType>  
</ValueAnnotation>

### The edm:Collection Expression

The 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 MUST contain zero or more child expressions. The values of the child expressions MUST all be type compatible. Each child expression MUST be a constant expression or a Record expression.

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

<ValueAnnotation Term="org.odata.seo.SeoTerms">  
 <Collection>  
 <String>Product</String>  
 <String>Supplier</String>  
 <String>Customer</String>  
 </Collection>  
</ValueAnnotation>

### The edm:EntitySetReference Expression

The value of an EntitySetReference is a reference to an entity set. A reference to an entity set is a collection of entities.

The EntitySetReference expression MUST contain a value of the type <qualifiedIdentifier>. The value of the entity set reference expression MUST resolve to an entity set.

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

<ValueAnnotation Term="org.odata.seo.SaleProducts">  
 <EntitySetReference>Self.SaleProducts</EntitySetReference>  
</ValueAnnotation>

### The edm:EnumMemberReference Expression

The value of an EnumMemberReference is a reference to a member of an enumeration type.

The EnumMemberReference expression MUST contain a value of the type <qualifiedIdentifier>. The value of the enum member reference expression MUST resolve to a member of an enumeration type.

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

<ValueAnnotation Term="org.odata.address.Type">  
 <EnumMemberReference>org.odata.address.Type.Mailing</EnumMemberReference>  
</ValueAnnotation>

### The edm:FunctionReference Expression

The value of a FunctionReference is a reference to the return type of a function.

The FunctionReference expression MUST contain a value of the type <qualifiedName>. The value of the function reference expression MUST resolve to a valid signature of a function.

The FunctionReference expression MUST contain zero or more expressions. The expressions contained within the function reference expression are used to determine which overload of the function is being referenced.

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

<ValueAnnotation Term="org.odata.person.Age">  
 <FunctionReference Function="org.odata.person.GetAge">  
 <Path>BirthDate</Path>  
 </FunctionReference>  
</ValueAnnotation>

### The edm:If Expression

The If expression enables a value to be obtained by evaluating a conditional expression.

An if expression MUST contain exactly three child 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. If the conditional expression evaluates to true, the second child expression MUST be evaluated and its value MUST be returned as the result of the 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 If expression.

The second child expression MUST return a value that is type compatible with the value returned from the third child expression.

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

<ValueAnnotation Term="org.odata.person.Gender">  
 <If>  
 <Path>IsFemale</Path>  
 <String>Female</String>  
 <String>Male</String>  
 </If>  
</ValueAnnotation>

### The edm:IsType Expression

The IsType expression evaluates a child expression and returns a boolean value indicating whether the child expression returns the specified type. The is type expression MUST assign a value of the type <anySingleTypeIdentifier> to the Type attribute.

An IsType expression MUST contain exactly one child expression. The is type expression MUST return true if the child expression returns a type that is compatible with the type named in the Type attribute. The is type expression MUST return false if the child expression returns a type that is not compatible with the type named in the Type attribute.

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

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

### The edm:LabeledElement Expression

The 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. The labeled element expression MUST assign a value of the type <simpleIdentifier> to the Name attribute.

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. The value of the child expression can also be accessed elsewhere by a labeled element reference expression.

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

<ValueAnnotation Term="org.odata.display.DisplayName">  
 <LabeledElement Name="CustomerFirstName">  
 <Path>FirstName</Path>  
 </LabeledElement>  
</ValueAnnotation>

### The edm:LabeledElementReference Expression

The LabeledElementReference expression returns the value of a labeled element expression.

The labeled element reference expression MUST contain a value of the type <simpleIdentifier>. The value of the <simpleIdentifier> MUST resolve to a labeled element expression.

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

<ValueAnnotation Term="org.odata.display.DisplayName">  
 <LabeledElementReference>DisplayName</LabeledElement>  
</ValueAnnotation>

### The edm:Null Expression

The 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, as shown in the following example:

<ValueAnnotation Term="org.odata.display.DisplayName">  
 <Null />  
</ValueAnnotation>

### The edm:ParameterReference Expression

### The edm:Path Expression

The Path expression enables a value to be obtained by traversing an object graph. The Path expression must be assigned a value of the type <pathString>.

The value assigned to the path expression MUST be composed of zero or more segments joined together by forward slashes. Each segment MUST represent a type cast, a property or a navigation property.

If the path segment represents a type cast, the segment MUST be of the type <qualifiedIdentifier>. If the path segment represents a property or a navigation property, the segment MUST be of the type <simpleIdentifier> and MUST resolve to a property or a navigation property of the same name.

If a path segment traverses a navigation property that has a cardinality of many, the path MUST NOT have any subsequent segments.

If the Path expression is an empty string or an empty element, the path MUST refer to the containing object.

The Path expression may be written with either element notation or attribute notation, as shown in the following examples:

<ValueAnnotation Term="org.odata.display.DisplayName" Path="FirstName" />  
<ValueAnnotation Term="org.odata.display.DisplayName">  
 <Path>FirstName</Path>  
</ValueAnnotation>

### The edm:PropertyReference Expression

The value of a PropertyReference is a reference to a structural property.

The PropertyReference expression MUST contain a value of the type <qualifiedName>. The value of the function reference expression MUST resolve to a valid signature of a function.

The FunctionReference expression MUST contain zero or more expressions. The expressions contained within the function reference expression are used to determine which overload of the function is being referenced.

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

<ValueAnnotation Term="org.odata.person.Age">  
 <FunctionReference Function="org.odata.person.GetAge">  
 <Path>BirthDate</Path>  
 </FunctionReference>  
</ValueAnnotation>

### The edm:Record Expression

The Record expression enables a new entity type or complex type to be constructed.

A record expression can specify a <simpleIdentifier> or <qualifiedIdentifier> value for the Type attribute. If no value is specified for the type attribute, the type is derived from the expression’s context. If a value is specified, the value MUST resolve to an entity type or complex type in the entity model.

A record expression MUST contain zero or more edm:PropertyValue elements.

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

<ValueAnnotation Term="org.odata.person.Employee">  
 <Record>  
 <PropertyValue Property="GivenName" Path="FirstName" />  
 <PropertyValue Property="Surname" Path="LastName" />  
 </Record>  
</ValueAnnotation>

### The edm:ValueTermReference Expression

# CSDL Examples

Something about these two examples.

## Title for example

<edmx:Edmx xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx" Version="1.0">

<edmx:DataServices xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"m:DataServiceVersion="2.0">

<Schema xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"xmlns="http://schemas.microsoft.com/ado/2007/05/edm" Namespace="ODataDemo">

<EntityType Name="Product">

<Key>

<PropertyRef Name="ID"/>

</Key>

<Property Name="ID" Type="Edm.Int32" Nullable="false"/>

<Property Name="Name" Type="Edm.String" Nullable="true" m:FC\_TargetPath="SyndicationTitle"m:FC\_ContentKind="text" m:FC\_KeepInContent="false"/>

<Property Name="Description" Type="Edm.String" Nullable="true" m:FC\_TargetPath="SyndicationSummary"m:FC\_ContentKind="text" m:FC\_KeepInContent="false"/>

<Property Name="ReleaseDate" Type="Edm.DateTime" Nullable="false"/>

<Property Name="DiscontinuedDate" Type="Edm.DateTime" Nullable="true"/>

<Property Name="Rating" Type="Edm.Int32" Nullable="false"/>

<Property Name="Price" Type="Edm.Decimal" Nullable="false"/>

<NavigationProperty Name="Category" Relationship="ODataDemo.Product\_Category\_Category\_Products"FromRole="Product\_Category" ToRole="Category\_Products"/>

<NavigationProperty Name="Supplier" Relationship="ODataDemo.Product\_Supplier\_Supplier\_Products"FromRole="Product\_Supplier" ToRole="Supplier\_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" m:FC\_TargetPath="SyndicationTitle"m:FC\_ContentKind="text" m:FC\_KeepInContent="true"/>

<NavigationProperty Name="Products" Relationship="ODataDemo.Product\_Category\_Category\_Products"FromRole="Category\_Products" ToRole="Product\_Category"/>

</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" m:FC\_TargetPath="SyndicationTitle"m:FC\_ContentKind="text" m:FC\_KeepInContent="true"/>

<Property Name="Address" Type="ODataDemo.Address" Nullable="false"/>

<Property Name="Concurrency" Type="Edm.Int32" Nullable="false" ConcurrencyMode="Fixed"/>

<NavigationProperty Name="Products" Relationship="ODataDemo.Product\_Supplier\_Supplier\_Products"FromRole="Supplier\_Products" ToRole="Product\_Supplier"/>

</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>

<Association Name="Product\_Category\_Category\_Products">

<End Role="Product\_Category" Type="ODataDemo.Product" Multiplicity="\*"/>

<End Role="Category\_Products" Type="ODataDemo.Category" Multiplicity="0..1"/>

</Association>

<Association Name="Product\_Supplier\_Supplier\_Products">

<End Role="Product\_Supplier" Type="ODataDemo.Product" Multiplicity="\*"/>

<End Role="Supplier\_Products" Type="ODataDemo.Supplier" Multiplicity="0..1"/>

</Association>

<EntityContainer Name="DemoService" m:IsDefaultEntityContainer="true">

<EntitySet Name="Products" EntityType="ODataDemo.Product"/>

<EntitySet Name="Categories" EntityType="ODataDemo.Category"/>

<EntitySet Name="Suppliers" EntityType="ODataDemo.Supplier"/>

<AssociationSet Name="Products\_Category\_Categories"Association="ODataDemo.Product\_Category\_Category\_Products">

<End Role="Product\_Category" EntitySet="Products"/>

<End Role="Category\_Products" EntitySet="Categories"/>

</AssociationSet>

<AssociationSet Name="Products\_Supplier\_Suppliers"Association="ODataDemo.Product\_Supplier\_Supplier\_Products">

<End Role="Product\_Supplier" EntitySet="Products"/>

<End Role="Supplier\_Products" EntitySet="Suppliers"/>

</AssociationSet>

<FunctionImport Name="GetProductsByRating" EntitySet="Products"ReturnType="Collection(ODataDemo.Product)" m:HttpMethod="GET">

<Parameter Name="rating" Type="Edm.Int32" Mode="In"/>

</FunctionImport>

</EntityContainer>

</Schema>

</edmx:DataServices>

</edmx:Edmx>

Example 2 (needs to be reviewed)

<Schema xmlns=" http://schemas.microsoft.com/ado/2009/11/edm " Namespace="Model1" Alias="Self">

  <Using Alias="Vocabulary1" Namespace="Vocabulary1" />

  <EntityContainer Name="Model1Container" >

    <EntitySet Name="CustomerSet" EntityType="Model1.Customer" />

    <EntitySet Name="OrderSet" EntityType="Model1.Order" />

    <AssociationSet Name="CustomerOrder" Association="Model1.CustomerOrder">

      <End Role="Customer" EntitySet="CustomerSet" />

      <End Role="Order" EntitySet="OrderSet" />

    </AssociationSet>

  </EntityContainer>

  <Annotations Target="Self.Customer">

    <ValueAnnotation Term="Vocabulary1.EMail">

      <Null />

    </ValueAnnotation>

    <ValueAnnotation Term="AccountID" Path="AccountNumber" />

    <ValueAnnotation Term="Title" String="Customer Info"/>

  </Annotations>

  <EntityType Name="Customer">

    <Key>

      <PropertyRef Name="CustomerId" />

    </Key>

    <Property Name="CustomerId" Type="Int32" Nullable="false" />

    <Property Name="FirstName" Type="String" Nullable="true" />

    <Property Name="LastName" Type="String" Nullable="true" />

    <Property Name="AccountNumber" Type="Int32" Nullable="true" />

    <Property Name="Address" Type="Self.Address" Nullable="false" />

    <NavigationProperty Name="Orders" Relationship="Model1.CustomerOrder" FromRole="Customer" ToRole="Order" />

    <TypeAnnotation Term="Vocabulary1.Person">

      <ValueAnnotation Term="DisplayName">

        <Apply Function="String.Concat">

          <Path>FirstName</Path>

          <Path>LastName</Path>

        </Apply>

      </ValueAnnotation>

    </TypeAnnotation>

  </EntityType>

  <EntityType Name="Order">

    <Key>

      <PropertyRef Name="OrderId" />

    </Key>

    <Property Name="OrderId" Type="Int32" Nullable="false" />

    <Property Name="OrderDate" Type="Int32" Nullable="true" />

    <Property Name="Description" Type=" String" Nullable="true" />

    <NavigationProperty Name="Customer" Relationship="Model1.CustomerOrder" FromRole="Order" ToRole="Customer" />

  </EntityType>

  <EntityType Name="SalesOrder" BaseType="Self.Order">

    <Property Name="Paid" Type="Boolean" Nullable="false" />

  </EntityType>

  <EntityType OpenType="true" Name="Product">

    <Key>

      <PropertyRef Name="ProductId" />

    </Key>

    <Property Name="ProductId" Type="Int32" Nullable="false" />

    <Property Name="Name" Type="String" Nullable="false" />

    <Property Name="Description" Type="String" Nullable="true" />

  </EntityType>

  <Association Name="CustomerOrder">

    <End Type="Model1.Customer" Role="Customer" Multiplicity="1" />

    <End Type="Model1.Order" Role="Order" Multiplicity="\*" />

  </Association>

  <ComplexType Name="Address">

    <Property Name="Street" Type="String" Nullable="false" />

    <Property Name="City" Type="String" Nullable="false" />

    <Property Name="State" Type="String" Nullable="false" />

    <Property Name="Zip" Type="String" Nullable="false" />

    <Property Name="Position" Type="GeographyPoint" Nullable="false" SRID="4326" />

  </ComplexType>

</Schema>

# ABNF for CSDL

identifierLeadingCharacter = ; Any character from the Unicode classes L or Nl

identifierCharacter = ; Any character from the Unicode classes L, Nl, Nd, Mn, Mc, Pc or Cf

simpleIdentifier = identifierLeadingCharacter \*identifierCharacter

qualifiedIdentifier = namespaceQualifiedIdentifier /  
 aliasQualifiedIdentifier

anyTypeReference = "collection(" anySingleTypeReference ")" /  
 anySingleTypeReference

anyKeylessTypeReference = "collection(" singleKeylessTypeReference ")" /  
 singleKeylessTypeReference

anySingleTypeReference = singleEntityTypeReference /  
 singleKeylessTypeReference

singleEntityTypeReference = qualifiedIdentifier / simpleIdentifier  
 ; identifies an EntityType by name

singleKeylessTypeReference = qualifiedIdentifier / simpleIdentifier  
 ; identifies a primitive, ComplexType or EnumType by name

namespaceSegment = simpleIdentifier

alias = simpleIdentifier

namespaceQualifiedIdentifier = namespace "." simpleIdentifier

aliasQualifiedIdentifier = alias "." simpleIdentifier

namespace = namespaceSegment \*("." namespaceSegment)

coreModel = "edm."

true = "true" / "1"

false = "false" / "0"

boolean = true / false

integralPrimitive = [coreModel] (  
 "byte" /  
 "int16" /  
 "int32" /  
 "int64" /  
 "sbyte"  
 )

spatialPrimitive = [coreModel] (  
 "geography" /  
 "geographypoint" /  
 "geographylinestring" /  
 "geographypolygon" /  
 "geographymultipoint" /  
 "geographymultilinestring" /  
 "geographymultipolygon" /  
 "geographycollection" /  
 "geometry" /  
 "geometrypoint" /  
 "geometrylinestring" /  
 "geometrypolygon" /  
 "geometrymultipoint" /  
 "geometrymultilinestring" /  
 "geometrymultipolygon" /  
 "geometrycollection" /  
 )

nonNegativeInt32 = 1\*10DIGIT ; 0-2147483647

primitive = ["edm."] (  
 integralPrimitive /  
 spatialPrimitive /  
 "" /  
 "" /  
 "" /  
 "" /  
 "" /  
 "" /  
 "" /  
 "" /  
 "" /  
 "" /  
 "" /  
 "" /  
 "" /  
 )