Common Schema Definition Language (CSDL) (OData Version 3.0)

OData Version 4.0 is the current recommended version of OData. OData V4 has been standardized by OASIS and has many features not included in OData Version 3.0.

➜ **Go to OData Version 4.0 (/documentation/)**

Introduction

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 [`XML 1.1 (https://www.w3.org/TR/2006/REC-xml11-20060816/)`] and [`XML Schema (https://www.w3.org/standards/techs/xmlschema#w3c_all)`].

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 `/$metadata`. `$metadata` MUST wrap the CSDL document in an EDMX wrapper.

1 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.

1.1 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.

1.2 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:

- CSDL version 1.0: http://schemas.microsoft.com/ado/2006/04/edm
- CSDL version 1.1: http://schemas.microsoft.com/ado/2007/05/edm
- CSDL version 1.2: http://schemas.microsoft.com/ado/2008/01/edm

- CSDL version 2.0: http://schemas.microsoft.com/ado/2008/09/edm
- CSDL version 3.0: http://schemas.microsoft.com/ado/2009/11/edm

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

1.3 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.

2 Common Characteristics of Entity Models

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

2.1 Nominal Types

A nominal type has a name. The name MUST be a [ ][csdl19]. In combination with a [csdl19] produces a fully qualified name of the form [ ][csdl19]. The [ ][csdl19] MUST be unique as it facilitates references to the element from other parts of the model.

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

- Fully qualified name
- Alias qualified name

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.example" Alias="example">
 <ComplexType Name="Address">...</ComplexType>
</Schema>
```

The various ways of referring to the nominal type are:

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

2.2 Structural Types

Structural types are composed of other model elements. Structural types are common in entity models as they are the typical means of representing entities in the OData service. Entity types and complex types are both structural types. Row types are less common but are also structural types.

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

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

2.3 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 [XML schema][csdl19] 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 MAY 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>
```

2.4 Vocabulary Annotations

Many parts of the model 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 Vocabulary Annotations for details on which model elements support vocabulary annotations.

2.5 Primitive Types

Structural types are composed of other structural types and primitive types. CSDL defines the following fully qualified primitive types:

- `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`
- `Edm.Stream`

3 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.

3.1 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. The data services element contains a description of the entity model(s) exposed by the OData service.

In addition to the data services element, `Edmx` may have zero or more `edmx:Reference` elements and zero or more `edmx:AnnotationsReference` elements. Reference elements specify the location of schemas referenced by the OData service. Annotations reference 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 `edmx: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>
```

3.1.1 The `Version` Attribute

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

The `Version` attribute is a string value that specifies the version of the EDMX wrapper, and must be of the form `.`. This version of the specification defines version `1.0` of the EDMX Wrapper.

3.2 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.

3.2.1 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: â€œ1.0â€, â€œ2.0â€, and â€œ3.0â€, corresponding to OData protocol versions 1.0, 2.0 and 3.0 respectively.

3.3 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 the reference element 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>
```

3.3.1 The `edmx:Url` Attribute

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

3.4 The `edmx:AnnotationsReference` Element

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

The annotations reference 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 the annotations reference element 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.example.validation" />
  <edmx:Include TermNamespace="org.example.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 the `org.example.validation` namespace
- Annotations that use a term from the `org.example.display` namespace and specify a `Slate` qualifier

3.4.1 The `edmx:Url` Attribute

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

3.5 The `edmx:Include` Element

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

3.5.1 The `edmx:TermNamespace` Attribute

An edmx:Include element MAY provide a [ ][csdl19] 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 include element will import the set of annotations that apply terms from the namespace in the value. The term namespace attribute also provides consumers insight about what namespaces are used in the annotations document. If there are no include elements that have a term namespace of interest to the consumer, the consumer can opt to not download the document.

3.5.2 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 include element will import the set of annotations that apply the qualifier in the value. The qualifier attribute also provides consumers insight about which qualifiers are used in the annotations document. If there are no include elements that have a qualifier of interest to the consumer, the consumer can opt to not download the document.

4 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 parts of the entity model.

4.1 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`

4.1.1 The `edm:Namespace` Attribute

A schema is identified by the value of the `edm:Namespace` attribute. The schemaâ€™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 Nominal Types for more detail.

A schema that contains nominal types MUST specify a [ ][csdl19] value for the namespace attribute. A schema that contains only vocabulary annotations MAY specify a [ ][csdl19] value for the namespace attribute. The `edm:Namespace` attribute MUST NOT use the reserved values `System`, `Transient` or `Edm`.

4.1.2 The `edm:Alias` Attribute

A schema MAY provide a [ ][csdl19] value for the `edm:Alias` attribute. An alias allows a CSDL document to qualify nominal types with a short string rather than a long namespace. For instance, `org.example.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 `edm: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 `edm:Using` element.

4.2 The `edm:Using` Element

The `edm: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 an entity type derived from an entity type in the importing model. In that case an `edm:EntitySet` in the importing model may return either entity type.

4.2.1 The `edm:Namespace` Attribute

A using element MUST provide a [ ][csdl19] value to the `edm:Namespace` attribute. The value provided to the namespace attribute SHOULD match the namespace of an entity model that is in scope.

4.2.2 The `edm:Alias` Attribute

A using element MUST define a [ ][csdl19] value for the `edm:Alias` attribute. An alias allows a CSDL model to substitute a short string for a long namespace. For instance, `org.example.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.

5 Properties

As mentioned in Structural Types, structural types are composed of other structural types and primitive types. Structural types 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" Defau
ltValue="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.

5.1 The `edm:Property` Element

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

5.1.1 The `edm:Name` Attribute

A property MUST specify a [ ][csdl19] value for the `edm: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.

5.2 The `edm:Type` Attribute

A property MUST specify a value for the `edm: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 [ ][csdl19]. The value of the type attribute MUST resolve to a complex type, enumeration type or primitive type, or a collection of complex, enumeration or primitive types.

5.3 Property Facets

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

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

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

In the following example the Nullable attribute MUST be placed on the child element that references the `DateTime` type. Facet attributes MUST NOT be applied to Collection type references.

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

5.3.1 The `edm:Nullable` Attribute

Any `edm:Property` MAY define a [ ][csdl19] value for the `edm:Nullable` facet attribute. The value of this attribute determines whether a value is required for the property on instances of the containing type.

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

5.3.2 The `edm:MaxLength` Attribute

A binary, stream or string `edm:Property` MAY define a [ ][csdl19] value for the `edm:MaxLength` facet attribute. The value of this attribute specifies the maximum length of the value of the property on a type instance.

5.3.3 The `edm:FixedLength` Attribute

A binary, stream or string `edm:Property` MAY define a [ ][csdl19] value for the `edm:FixedLength` facet attribute. The value of this attribute specifies the size of the array used to store the value of the property on a type instance.

5.3.4 The `edm:Precision` Attribute

A temporal or decimal `edm:Property` MAY define a [ ][csdl19] value for the `edm:Precision` attribute.

For a decimal property the value of this attribute specifies the maximum number of digits allowed in the propertyâ€™s value. For a temporal property the value of this attribute specifies the number of decimal places allowed in the seconds portion of the propertyâ€™s value.

5.3.5 The `edm:Scale` Attribute

A decimal `edm:Property` MAY define a [ ][csdl19] value for the `edm:Scale` attribute. The value of this attribute specifies the maximum number of digits allowed to the right of the decimal point.

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

5.3.6 The `edm:Unicode` Attribute

A string property MAY define a [ ][csdl19] value for the `edm: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` .

5.3.7 The `edm:Collation` Attribute

A string property MAY define a value for the `edm: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`

5.3.8 The `edm:SRID` Attribute

A spatial property MAY define a value for the `edm: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 [ ][csdl19] 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)][http://www.epsg.org/Geodetic.html].

5.3.9 The `edm:DefaultValue` Attribute

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

5.3.10 The `edm:ConcurrencyMode` Attribute

An `edm:Property` MAY define a value for the `edm: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.

6 Entity Type Constructs

Entity types are nominal structural types 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="Self.Product_Category_Category_P
roducts" FromRole="Product_Category" ToRole="Category_Products"/>
 <NavigationProperty Name="Supplier" Relationship="Self.Product_Supplier_Supplier_P
roducts" FromRole="Product_Supplier" ToRole="Supplier_Products"/>
</EntityType>
```

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

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

6.1 The `edm:EntityType` Element

The `edm: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 entity type element also can contain zero or more `edm:NavigationProperty` elements.

6.1.1 The `edm:Name` Attribute

A value of the form [ ][csdl19] MUST be provided for the `edm:Name` attribute because an entity type is a nominal type. The value identifies the entity type and MUST be unique within the entity typeâ€™s namespace.

6.1.2 The `edm:BaseType` Attribute

An entity type can inherit from another entity type by specifying a [ ][csdl19] value for the `edm: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 typeâ€™s base type.

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

6.1.3 The `edm:Abstract` Attribute

An entity type MAY indicate that it cannot be instantiated by providing a [ ][csdl19] value of `true` to the `edm:Abstract` attribute. If not specified, the abstract attribute defaults to `false`.

6.1.4 The `edm:OpenType` Attribute

An entity type MAY indicate that it can be freely extended by providing a [ ][csdl19] value of `true` to the `edm: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.

6.1.5 The `metadata:HasStream` Attribute

An entity type MAY contain the `metadata:hasstream` attribute.

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

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

6.2 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 `edm:Key` element. An entity typeâ€™s key refers to the set of properties that uniquely identify an instance of the entity type.

If specified, the key MUST contain one or more `edm:PropertyRef` elements. An `edm:PropertyRef` element references an `edm:Property`. The properties that compose the key MUST be non-nullable primitives or enumeration types.

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>
```

6.3 The `edm:PropertyRef` Element

The `edm:PropertyRef` element provides an `edm:Key` with a reference to a single property of an entity type. A [ ][csdl19] value MUST be supplied to the `edm:Name` attribute. This value MUST resolve to one of the properties of the entity type.

6.4 The `edm:NavigationProperty` Element

A navigation property allows navigation from an entity to related entities.

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

```
<EntityType Name="Product">
 ...
 <NavigationProperty Name="Category" ToRole="Category" FromRole="Product" Relations
hip="Self.ProductCategory"/>
 <NavigationProperty Name="Supplier" ToRole="Supplier" FromRole="Product" Relations
hip="Self.ProductSupplier"/>
</EntityType>
<EntityType Name="Category">
 ...
 <NavigationProperty Name="Products" ToRole="Product" FromRole="Category" Relations
hip="Self.ProductCategory"/>
</EntityType>
```

6.4.1 The `edm:Name` Attribute

The navigation property MUST provide a [ ][csdl19] value to the `edm: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.

6.4.2 The `edm:Relationship` Attribute

The `edm:Relationship` attribute MUST be given a [ ][csdl19] or [ ][csdl19] value. The value of the attribute MUST resolve to an association in the entity model.

6.4.3 The `edm:ToRole` Attribute

The navigation property MUST provide a [ ][csdl19] value to the `edm:ToRole` attribute. The `edm:ToRole` attribute is a name used to refer to the destination of the navigation property.

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

6.4.4 The `edm:FromRole` Attribute

The navigation property MUST provide a [ ][csdl19] value to the `edm:FromRole` attribute. The `edm:FromRole` attribute is a name used to refer to the destination of the navigation property.

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

6.4.5 The `edm:ContainsTarget` Attribute

A [ ][csdl19] value MAY be assigned to the `edm:ContainsTarget` attribute. If no value is assigned to the contains target attribute, the attribute defaults to `false`. If the value assigned to the `edm:ContainsTarget` attribute is `true`, the entity type to which the navigation property belongs is said to contain the destination of the navigation property.

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

When a navigation property navigates between entity types in the same entity set it is called recursive containment. If the containment is recursive, the containing entity type MUST have a multiplicity of `0..1`. If the containment is not recursive, the containing entity type MUST have a multiplicity of `1`.

The association set for a containment navigation property MUST specify the same entity set for entity types with a common ancestor in the inheritance chain.

An entity set MUST NOT be bound by association set to more than one navigation property via a containment navigation property that indicates that the entity type (or derived entity types) of that entity set is contained.

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

7.1 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 `edm:Name` attribute as well as one or more `edm:Property` elements. Complex types MUST NOT have any navigation properties.

8 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 `edm: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>
```

8.1 The `edm:EnumType` Element

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

The enumeration type MUST provide a [ ][csdl19] as the value of the `edm:Name` attribute.

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

8.1.1 The `edm:UnderlyingType` Attribute

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

The enumeration type MUST assign an value to the `edm:UnderlyingType` attribute. If the underlying type is not specified, a 32-bit integer MUST be used as the underlying type.

8.1.2 The `edm:IsFlags` Attribute

An enumeration type MAY specify a [ ][csdl19] value for the `edm:IsFlags` attribute. A value of `true` indicates that the enumeration type allows multiple members to be selected simultaneously.

8.2 The `edm:Member` Element

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

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

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

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

8.2.1 The `edm:Name` Attribute

Each enumeration member MUST provide a [ ][csdl19] value for the `edm:Name` attribute. The enumeration type MUST NOT declare two members with the same name.

8.2.2 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.

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

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

`Overnight` a value of `5`.

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

9 Other Type Constructs

9.1 Collection Types

A collection type provides a means of representing 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.

9.1.1 The `edm:CollectionType` Element

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

The collection type element can identify the types contained in the collection by specifying a [ ][csdl19] or [ ][csdl19] value for the `edm: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.

9.2 The `edm:TypeRef` Element

The `edm:TypeRef` element is used to reference a nominal type.

The `edm:TypeRef` element MUST provide a [ ][csdl19] or [ ][csdl19] value for the `edm:Type` attribute.

The type ref can define relevant facets for scalar types.

9.3 Reference Types

A reference type specifies a reference to an entity instance. A reference to an entity instance is a special structure consisting primarily of the entity instance key. 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>
```

9.3.1 The `edm:ReferenceType` Element

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

A reference type MUST specify a [ ][csdl19] value for the `edm:Type` attribute. The value of this attribute names the type for which the reference type contains key information.

9.4 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 importâ€™s return type.

9.4.1 The `edm:RowType` Element

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

10 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="Self.Product" Multiplicity="*" Role="Product"/>
 <End Type="Self.Category" Multiplicity="0..1" Role="Category"/>
</Association>
```

10.1 The `edm:Association` Element

The `edm:Association` element represents an association in an entity model. The association MUST provide a [ ][csdl19] value for the `edm:Name` attribute.

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

10.2 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 `edm:End` element describes one of the ends of an association.

Each end MAY also contain zero or one `edm:OnDelete` elements to specify operational behavior.

10.2.1 The `edm:Type` Attribute

Each end of the association MUST specify the entity type attached to that end. The value of the `edm:Type` attribute must be a [ ][csdl19]. The value of the type attribute MUST resolve to an entity type in the entity model.

10.2.2 The `edm:Role` Attribute

The `edm:Role` attribute allows the association end to be bound to a navigation property. The association end MAY assign a [ ][csdl19] value to the `edm:Role` attribute.

10.2.3 The `edm:Multiplicity` Attribute

The `edm:Multiplicity` attribute defines the cardinality of the association end. The value of the attribute MUST be one of the following: - `0..1` â€“ zero or one - `1` â€“ exactly one - `*` â€“ zero or more

10.3 The `edm:OnDelete` Element

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

If present, the `edm:OnDelete` element MUST define a value for the `edm:Action` attribute. The value assigned to the action attribute MUST be `Cascade` or `None`.

10.4 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 `edm: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="Self.Product" Multiplicity="*" Role="Product"/>
 <End Type="Self.Category" Multiplicity="0..1" Role="Category"/>
 <ReferentialConstraint>
  <Principal Role="Category">
   <PropertyRef Name="CategoryID" />
  </Principal>
  <Dependent Role="Product">
   <PropertyRef Name="CategoryID" />
  </Dependent>
 </ReferentialConstraint>
</Association>
```

10.5 The `edm:Principal` Element

The `edm: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 [ ][csdl19] value for the `edm:Role` attribute. This value MUST have the same value as the `edm: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.

10.6 The `edm:Dependent` Element

The `edm: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 dependent element MUST provide a [ ][csdl19] value for the `edm:Role` attribute. This value MUST have the same value as the `edm:Role` attribute of one of the association ends.

10.7 The `edm:PropertyRef` Element

The `edm: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 `edm:PropertyRef` elements. The property references MUST each be assigned [ ][csdl19] 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="Self.OrderLineDetail" Multiplicity="*" Role="Detail"/>
 <End Type="Self.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>
```

11 Model Functions

Model functions are similar to function imports in that they represent a function that can interact with types in the model. Model functions allow the definition of a function body directly in the model. This specification does not place any requirements on the content of the function body.

For example, the following function represents a means of calculating an age:

```
<Function Name="GetAge" ReturnType="Edm.Int32">
 <Parameter Name="Person" Type="Self.Person" />
 <DefiningExpression>
  Edm.DiffYears(Edm.CurrentDateTime(), Person.Birthday)
 </DefiningExpression>
</Function>
```

In the preceding example the function defines a function body using [Entity SQL][https://msdn.microsoft.com/en-us/library/bb387145.aspx]. This example is informative only; it is up to a given runtime to determine the validity of the defining expression.

11.1 The `edm:Function` Element

The `edm:Function` element is a nominal type that represents a user function in an entity model. The function may return a single or collection of primitive types, row types or nominal structured types.

A function MUST provide a [ ][csdl19] value for the `edm:Name` attribute.

The function MUST specify exactly one return type. The return type can be specified using the `edm:ReturnType` attribute or the `edm:ReturnType` element.

The function MAY also contain zero or more `edm:Parameter` elements to be used during the execution of the function.

All parameters and the return type MUST be one of the following:

- A scalar type or a collection of scalar types
- An entity type or a collection of entity types
- A complex type or a collection of complex types
- A row type or a collection of row types
- A reference type or a collection of reference types

The function MAY contain zero or one `edm:DefiningExpression` elements that state the definition for the function body. The conceptual schema definition language (CSDL) file format does not specify rules and restrictions regarding what language is to be used for expressing function bodies.

11.1.1 The `edm:ReturnType` Attribute

If the return type is written with attribute notation, an [ ][csdl19] value MUST be provided for the `edm:ReturnType` attribute.

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

11.2 The `edm:Parameter` Element

The `edm:Parameter` element represents a parameter to the function. A [ ][csdl19] value MUST be provided for the `edm:Name` attribute.

The parameter element MUST specify exactly one type. The type may be specified by providing an [ ][csdl19] value to the `edm:Type` attribute. Alternatively the type may be specified by providing at most one `edm:CollectionType` , `edm:ReferenceType` , or `edm:RowType` element as a child of the parameter element.

The parameter element MUST NOT set a value for the `edm:Mode` attribute. The mode for function parameters is always `In` .

11.3 The `edm:ReturnType` Element

If the return type is written with element notation, the `edm:Function` MUST contain exactly one `edm:ReturnType` element.

The return type element MUST specify exactly one type. The type may be specified by providing a [ ][csdl19] value to the `edm:Type` attribute. Alternatively the type may be specified by providing at most one `edm:CollectionType` , `edm:ReferenceType` , or `edm:RowType` element as a child of the return type element.

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

12 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 type that allows access to entity instances. Simple entity models frequently have one entity set per entity type, for example:

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

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

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

In this case the `Products` entity set could expose products that have not been discontinued and the `DiscontinuedProducts` entity set could expose products that have been discontinued.

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 type that disambiguates the entity set used by an association end. In the preceding example an `Self.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 `Self.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="Col
lection(Self.Product)">
 <Parameter Name="fiscalYear" Mode="In" Type="Edm.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="Self.Product"/>
 <EntitySet Name="Categories" EntityType="Self.Category"/>
 <EntitySet Name="Suppliers" EntityType="Self.Supplier"/>
 <AssociationSet Name="ProductCategorySet" Association="Self.ProductCategory">
  <End Role="Product" EntitySet="Products"/>
  <End Role="Category" EntitySet="Categories"/>
 </AssociationSet>
 <AssociationSet Name="ProductSupplierSet" Association="Self.ProductSupplier">
  <End Role="Product" EntitySet="Products"/>
  <End Role="Supplier" EntitySet="Suppliers"/>
 </AssociationSet>
 <FunctionImport Name="GetProductsByRating" EntitySet="Products" ReturnType="Collec
tion(Self.Product)">
  <Parameter Name="rating" Type="Edm.Int32" Mode="In"/>
 </FunctionImport>
</EntityContainer>
```

12.1 The `edm:EntityContainer` Element

The `edm:EntityContainer` element represents an entity container in an entity model. It corresponds to a logical data store and contains zero or more `edm:EntitySet`, `edm:AssociationSet` or `edm:FunctionImport` elements.

The entity container MUST provide a unique [ ][csdl19] value for the `edm:Name` attribute.

An entity container MAY provide a [ ][csdl19] value for the `edm: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.

12.2 The `edm:EntitySet` Element

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

An entity set MUST provide a [ ][csdl19] value for the `edm:Name` attribute. An entity set also has an `edm:EntityType` attribute that MUST be provided with a [ ][csdl19] 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 `edm:EntityType` attribute or its subtypes. The entity type named by the entity type attribute MAY be abstract.

12.3 The `edm:AssociationSet` Element

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

An association set MUST provide a [ ][csdl19] value for the `edm:Name` attribute. An association set also has an `edm:Association` attribute that MUST be provided with a [ ][csdl19] that resolves to an association in the entity model.

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

12.3.1 The `edm:End` Element

An `edm: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 [ ][csdl19] value for the `edm:Role` attribute that is the same as the value of one of the association endsâ€™ `edm:Role` attribute.

The `edm:End` element MUST also provide a [ ][csdl19] or [ ][csdl19] value for the `edm:EntitySet` attribute. The entity set that is named MUST expose the entity type bound by the corresponding end of the association.

12.4 The `edm:FunctionImport` Element

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

Functions MUST NOT have observable side-effects and MUST return a single or collection of primitive types or nominal structured types. Functions MAY be composable.

Actions MAY have side-effects and MAY return a single or collection of primitive types or nominal structured types. Actions are not composable.

A function import MUST provide a [ ][csdl19] value for the `edm:Name` attribute.

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

If the function import specifies a return type that is an entity or a collection of entities, it MUST include either an `edm:EntitySet` Attribute or an `edm:EntitySetPath` Attribute to specify the entityset of the returned entity or collection of entities.

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

12.4.1 The `edm:ReturnType` Attribute

If the return type is written with attribute notation, an [ ][csdl19] value MUST be provided for the `edm:ReturnType` attribute.

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

12.4.2 The `edm:EntitySet` Attribute

If the return type is an entity or a collection of entities, a [ ][csdl19] value MAY 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.

12.4.3 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 importâ€™s parameters. The remaining segments of the entity set path MUST represent navigation or type casts.

A navigation segment simply names the [ ][csdl19] of the navigation property to be traversed. A type cast segment names the [ ][csdl19] of the entity type that should be returned from the type cast.

12.4.4 The `edm:IsSideEffecting` Attribute

A function import that may have side effects represents an *Action*. A function import that does not have side effects represents a *Function*.

The function import MAY specify a [ ][csdl19] value for the `edm:IsSideEffecting` attribute. If no value is specified for the is side effecting attribute, the value defaults to `true`, indicating that the function import represents an Action.

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

12.4.5 The `edm:IsBindable` Attribute

A function import can specify a [ ][csdl19] value for the `edm:IsBindable` attribute. If no value is specified for the `edm:IsBindable` attribute, the value defaults to `false`.

If the is bindable attribute is set to `true`, the function import MUST contain at least one `edm:Parameter` element, and the first such parameter must represent an entity or collection of entities.

12.4.6 The `edm:IsComposable` Attribute

A function import can specify a [ ][csdl19] value for the `edm:IsComposable` attribute. If no value is specified for the is composable attribute, the value defaults to `false`.

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

12.5 The `edm:ReturnType` Element

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

If element notation is used, a similar set of attributes can be used to specify the return type of the function. The `edm: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.

12.6 The `edm:Parameter` Element

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

The `edm:Name` attribute MUST be used to assign a unique [ ][csdl19] value to the parameter.

12.6.1 The `edm:Type` Attribute

The parameter MUST indicate which set of types can be passed to the parameter by providing a [ ][csdl19] value for the `edm:Type` attribute.

12.6.2 The `edm:Mode` Attribute

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

12.6.3 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.

13 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.

14 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.

14.1 `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.

14.2 The `edm:ValueTerm` Element

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

The value term MUST provide a [ ][csdl19] value for the `edm:Name` attribute and a [ ][csdl19] or [ ][csdl19] value for the `edm: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.

15 Vocabulary Annotations

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

- `edm:Annotations`
- `edm:ComplexType`
- `edm:EntitySet`
- `edm:EntityType`
- `edm:EnumType`
- `edm:Function`
- `edm:FunctionImport`
- `edm:NavigationProperty`
- `edm:Parameter`
- `edm:Property`
- `edm: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 parts of the model, for example: 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.

15.1 The `edm:Annotations` Element

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

An annotations element MUST assign a [ ][csdl19] or [ ][csdl19] value to the `edm:Target` attribute. The value of the target attribute SHOULD resolve to a model element in the entity model.

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

15.2 The `edm:TypeAnnotation` Element

The `edm:TypeAnnotation` element represents a type annotation. A type annotation binds a type term to a part of the entity model.

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

A type annotation MUST provide a [ ][csdl19] or [ ][csdl19] value for the `edm:Term` attribute. The value of the term attribute SHOULD resolve to an entity type or complex type.

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

15.3 The `edm:PropertyValue` Element

The `edm: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 [ ][csdl19] value to the `edm: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 MAY be provided using element notation or attribute notation.

15.4 The `edm:ValueAnnotation` Element

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

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

A value annotation MUST provide a [ ][csdl19] or [ ][csdl19] value for the `edm: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 MAY be provided using element notation or attribute notation.

15.5 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.example.display.DisplayName" Path="FirstName" Qualifier
="org.example.formfactor.Slate">
```

The value of the `edm: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 annotations element MUST NOT provide a value for the qualifier attribute.

16 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.example.display.DisplayName">
 <String>Customers</String>
</ValueAnnotation>
```

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

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

16.1 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="Self.Product">
 <ValueAnnotation Term="org.example.display.DisplayName" String="Product Catalog" /
>
</EntitySet>

<EntitySet Name="Suppliers" EntityType="Self.Supplier">
 <ValueAnnotation Term="org.example.display.DisplayName" String="Supplier Director
y" />
</EntitySet>
```

16.1.1 The `edm:Binary` Constant Expression

The `edm:Binary` constant expression evaluates to a primitive binary value. A binary expression MUST be assigned a value of the type [ ][csdl19].

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

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

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

16.1.2 The `edm:Bool` Constant Expression

The `edm:Bool` constant expression evaluates to a primitive boolean value. A boolean expression MUST be assigned a value of the type [ ][csdl19].

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

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

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

16.1.3 The `edm:DateTime` Constant Expression

The `edm:DateTime` constant expression evaluates to a primitive date/time value. A date/time expression MUST be assigned a value of the type [ ][csdl19]. 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.example.display.LastUpdated" DateTime="2000-01-01T16:00:00.000" />

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

16.1.4 The `edm:DateTimeOffset` Constant Expression

The `edm:DateTimeOffset` constant expression evaluates to a primitive date/time value with a UTC offset. An `edm:DateTimeOffset` expression MUST be assigned a value of the type [ ][csdl19].

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

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

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

16.1.5 The `edm:Decimal` Constant Expression

The `edm:Decimal` constant expression evaluates to a primitive decimal value. A decimal expression MUST be assigned a value of the type [ ][csdl19].

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

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

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

16.1.6 The `edm:Float` Constant Expression

The `edm:Float` constant expression evaluates to a primitive floating point (or double) value. A floating point expression MUST be assigned a value of the type [ ][csdl19].

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

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

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

16.1.7 The `edm:Guid` Constant Expression

The `edm:Guid` constant expression evaluates to a primitive 32-character string value. A guid expression MUST be assigned a value of the type [ ][csdl19].

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

```
<ValueAnnotation Term="org.example.display.Id" Guid="21EC2020-3AEA-1069-A2DD-08002B
30309D" />

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

16.1.8 The `edm:Int` Constant Expression

The `edm:Int` constant expression evaluates to a primitive integral value. An integral expression MUST be assigned a value of the type [ ][csdl19].

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

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

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

16.1.9 The `edm:String` Constant Expression

The `edm:String` constant expression evaluates to a primitive string value. A string expression MUST be assigned a value of the type [ ][csdl19].

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

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

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

16.1.10 The `edm:Time` Constant Expression

The `edm: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 [ ][csdl19].

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

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

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

16.2 Dynamic Expressions

16.2.1 The `edm:Apply` Expression

The `edm:Apply` expression enables a value to be obtained by applying a client-side function. An apply expression MUST assign a string value to the `edm:Function` attribute. The value of the function attribute SHOULD be a [ ][csdl19]. The value of the function attribute is used to locate the client-side function that should be applied.

The `edm: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.example.display.DisplayName">
 <Apply Function="org.example.commonfunctions.StringConcat">
 <String>Product</String>
 <String> </String>
 <String>Catalog</String>
 </Apply>
</ValueAnnotation>
```

16.2.2 The `edm:AssertType` Expression

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

The assert type expression MUST assign a value of the type [ ][csdl19] to the `edm:Type` attribute.

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

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

16.2.3 The `edm:Collection` Expression

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

A collection expression 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 `edm:Record` expression.

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

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

16.2.4 The `edm:EntitySetReference` Expression

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

The `edm:EntitySetReference` expression MUST contain a value of the type [ ][csdl19]. The value of the entity set reference expression MUST resolve to an entity set.

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

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

16.2.5 The `edm:EnumMemberReference` Expression

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

The `edm:EnumMemberReference` expression MUST contain a value of the type [ ][csdl19]. The value of the enum member reference expression MUST resolve to a member of an enumeration type.

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

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

16.2.6 The `edm:FunctionReference` Expression

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

The `edm:FunctionReference` expression MUST contain a value of the type [ ][csdl19]. The value of the function reference expression MUST resolve to a valid signature of a function.

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

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

16.2.7 The `edm:If` Expression

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

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

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

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

16.2.8 The `edm:IsType` Expression

The `edm: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 [ ][csdl19] to the `edm:Type` attribute.

An `edm: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 `edm:Type` attribute.

The `edm: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>
```

16.2.9 The `edm:LabeledElement` Expression

The `edm:LabeledElement` expression assigns a name to a child expression. The value of the child expression can then be reused elsewhere with an `edm:LabeledElementReference` expression. The labeled element expression MUST assign a value of the type [ ][csdl19] to the `edm: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.example.display.DisplayName">
 <LabeledElement Name="CustomerFirstName">
  <Path>FirstName</Path>
 </LabeledElement>
</ValueAnnotation>
```

16.2.10 The `edm:LabeledElementReference` Expression

The `edm:LabeledElementReference` expression returns the value of a labeled element expression.

The labeled element reference expression MUST contain a value of the type [ ][csdl19]. The value of the [ ][csdl19] MUST resolve to a labeled element expression.

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

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

16.2.11 The `edm:Null` Expression

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

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

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

16.2.12 The `edm:ParameterReference` Expression

The value of a `edm:ParameterReference` expression is a reference to a function parameter.

The `edm:ParameterReference` expression MUST contain a value of the type [ ][csdl19]. The value of the parameter reference expression MUST resolve to a parameter.

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

```
<ValueAnnotation Term="org.example.base.IsUrl">
 <ParameterReference>Image</ParameterReference>
</ValueAnnotation>
```

16.2.13 The `edm:Path` Expression

The `edm:Path` expression enables a value to be obtained by traversing an object graph. The `edm:Path` expression must be assigned a value of the type [ ][csdl19].

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 [ ][csdl19]. If the path segment represents a property or a navigation property, the segment MUST be of the type [ ][csdl19] 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 `edm:Path` expression is an empty string or an empty element, the path MUST refer to the containing object.

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

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

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

16.2.14 The `edm:PropertyReference` Expression

The value of a `edm:PropertyReference` is a reference to a structural property.

The `edm:PropertyReference` expression MUST contain a value of the type [ ][csdl19]. The value of the property reference expression MUST resolve to a valid signature of a property.

```
<ValueAnnotation Term="org.example.AlternateKey">
 <PropertyReference Property="SSN"/>
</ValueAnnotation>
```

16.2.15 The `edm:Record` Expression

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

A record expression can specify a [ ][csdl19] or [ ][csdl19] value for the `edm:Type` attribute. If no value is specified for the type attribute, the type is derived from the expressionâ€™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.example.person.Employee">
 <Record>
  <PropertyValue Property="GivenName" Path="FirstName" />
  <PropertyValue Property="Surname" Path="LastName" />
 </Record>
</ValueAnnotation>
```

16.2.16 The `edm:ValueTermReference` Expression

The value of a `edm:ValueTermReference` expression is a reference to a value term.

The `edm:ValueTermReference` expression MUST contain a value of the type [ ][csdl19]. The value of the value term reference expression MUST resolve to a valid signature of a value term.

```
<ValueAnnotation Term="org.example.validation.IsRequired">
    <ValueTermReference Term="org.example.display.DisplayName" Qualifier="Mobile"/>
</ValueAnnotation>
```

17 CSDL Examples

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

17.1 Products and Categories Example

```
<edmx:Edmx xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx" Version
="1.0">
  <edmx:DataServices xmlns:metadata="http://schemas.microsoft.com/ado/2007/08/datas
ervices/metadata metadata:DataServiceVersion="3.0">
    <Schema xmlns="http://schemas.microsoft.com/ado/2007/05/edm" Namespace="ODataDe
mo">
      <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"/>
        <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_Categor
y_Category_Products"FromRole="Product_Category" ToRole="Category_Products"/>
        <NavigationProperty Name="Supplier" Relationship="ODataDemo.Product_Supplie
r_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"/>
        <NavigationProperty Name="Products" Relationship="ODataDemo.Product_Categor
y_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"/>
        <Property Name="Address" Type="ODataDemo.Address" Nullable="false"/>
        <Property Name="Concurrency" Type="Edm.Int32" Nullable="false" ConcurrencyM
ode="Fixed"/>
        <NavigationProperty Name="Products" Relationship="ODataDemo.Product_Supplie
r_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.P
roduct_Category_Category_Products">
          <End Role="Product_Category" EntitySet="Products"/>
          <End Role="Category_Products" EntitySet="Categories"/>
        </AssociationSet>
        <AssociationSet Name="Products_Supplier_Suppliers"Association="ODataDemo.Pr
oduct_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)">
          <Parameter Name="rating" Type="Edm.Int32" Mode="In"/>
        </FunctionImport>
      </EntityContainer>
    </Schema>
  </edmx:DataServices>
</edmx:Edmx>
```

17.2 Annotated Customers and Orders Example

```
<Schema xmlns="http://schemas.microsoft.com/ado/2009/11/edm " Namespace="Model1" Al
ias="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="Edm.Int32" Nullable="false" />
    <Property Name="FirstName" Type="Edm.String" Nullable="true" />
    <Property Name="LastName" Type="Edm.String" Nullable="true" />
    <Property Name="AccountNumber" Type="Edm.Int32" Nullable="true" />
    <Property Name="Address" Type="Self.Address" Nullable="false" />
    <NavigationProperty Name="Orders" Relationship="Model1.CustomerOrder" FromRole
="Customer" ToRole="Order" />
    <TypeAnnotation Term="Vocabulary1.Person">
      <PropertyValue Property="DisplayName">
        <Apply Function="String.Concat">
          <Path>FirstName</Path>
          <Path>LastName</Path>
        </Apply>
      </PropertyValue>
    </TypeAnnotation>
  </EntityType>
  <EntityType Name="Order">
    <Key>
      <PropertyRef Name="OrderId" />
    </Key>
    <Property Name="OrderId" Type="Edm.Int32" Nullable="false" />
    <Property Name="OrderDate" Type="Edm.Int32" Nullable="true" />
    <Property Name="Description" Type="Edm.String" Nullable="true" />
    <NavigationProperty Name="Customer" Relationship="Model1.CustomerOrder" FromRol
e="Order" ToRole="Customer" />
  </EntityType>
  <EntityType Name="SalesOrder" BaseType="Self.Order">
    <Property Name="Paid" Type="Edm.Boolean" Nullable="false" />
  </EntityType>
  <EntityType OpenType="true" Name="Product">
    <Key>
      <PropertyRef Name="ProductId" />
    </Key>
    <Property Name="ProductId" Type="Edm.Int32" Nullable="false" />
    <Property Name="Name" Type="Edm.String" Nullable="false" />
    <Property Name="Description" Type="Edm.String" Nullable="true" />
  </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="Edm.String" Nullable="false" />
    <Property Name="City" Type="Edm.String" Nullable="false" />
    <Property Name="State" Type="Edm.String" Nullable="false" />
    <Property Name="Zip" Type="Edm.String" Nullable="false" />
    <Property Name="Position" Type="Edm.GeographyPoint" Nullable="false" SRID="432
6" />
  </ComplexType>
</Schema>
```

18 Informative XSD for CSDL

```
Common Data Model Schema Definition Language.
Copyright (c) Microsoft Corp. All rights reserved.
```

```
The Documentation element is used to provide documentation of comments on the c
```

```
      This type allows pretty much any content
```