[MC-CSDL]:
Conceptual Schema Definition File Format

Revision Summary

Date	Revision History	Revision Class	Comments
02/27/2009	0.1	Major	First Release.
04/10/2009	0.1.1	Editorial	Revised and edited the technical content.
05/22/2009	0.1.2	Editorial	Revised and edited the technical content.
07/02/2009	0.2	Minor	Updated the technical content.
08/14/2009	0.2.1	Editorial	Revised and edited the technical content.
09/25/2009	1.0	Major	Updated and revised the technical content.
11/06/2009	1.0.1	Editorial	Revised and edited the technical content.
12/18/2009	1.0.2	Editorial	Revised and edited the technical content.
01/29/2010	1.1	Minor	Updated the technical content.
03/12/2010	1.1.1	Editorial	Revised and edited the technical content.
04/23/2010	1.1.2	Editorial	Revised and edited the technical content.
06/04/2010	1.1.3	Editorial	Revised and edited the technical content.
07/16/2010	1.2	Minor	Clarified the meaning of the technical content.
08/27/2010	1.3	Minor	Clarified the meaning of the technical content.
10/08/2010	2.0	Major	Significantly changed the technical content.
11/19/2010	3.0	Major	Significantly changed the technical content.
01/07/2011	4.0	Major	Significantly changed the technical content.
02/11/2011	4.1	Minor	Clarified the meaning of the technical content.

Contents

1 Introduction

This document describes the structure and semantics of the Conceptual Schema Definition Language (CSDL) for the Entity Data Model (EDM). CSDL Version 2.0 is a language based on XML that can be used for defining EDM-based conceptual models.

The EDM is an entity-relationship (ER) model. The ER model has existed for over 30 years and differs from the more familiar relational model, because Associations and Entities are all first-class concepts.

The EDM defines some well-known primitive types, such as Edm.String, which are used as the building blocks for structural types like Entity Types and Complex Types.

Entities are instances of Entity Types (for example, Customer or Employee) which are richly structured records with a key. The structure of an Entity Type is provided by its Properties. An entity key is formed from a subset of the properties of the Entity Type. The key (for example, CustomerId or EmployeeId) is a fundamental concept to uniquely identify and persist entity instances and to allow entity instances to participate in relationships or associations.

Entities are grouped in Entity Sets; for example, the EntitySet Customers is a set of Customer instances.

Associations (occasionally referred to as Relationships) are instances of Association Types. Association Types are used to specify a named relationship between two Entity Types. Thus, an Association is a named relationship between two or more Entities. Associations are grouped into Association Sets.

Entity Types may include one or more Navigation Properties. A Navigation Property is tied to an Association Type and allows the navigation from one end of an Association, the Entity Type that declares the Navigation Property, to the other related end, which can be anything from 0 or more related Entities. Unlike standard Properties, Navigation Properties are not considered to be structurally part of an Entity.

Complex Types, which are structural types similar to an Entity Type, are also supported by the EDM. The main difference is that Complex Types have no identity and can't support Associations. For these reasons, Complex Types instances only exist as properties of Entity Types (or other Complex Types).

The EDM also supports Entity Type and Complex Type inheritance.

Inheritance is a fundamental modeling concept that allows different types to be related in an "Is a" relationship enabling extensibility and reuse of existing entity types. When type B inherits from type A, then A is the base-type of B, and B is a sub-type or derived-type of A. The derived-type inherits all properties of its base-type and these properties are called inherited-properties. The derived-type can be extended to have more properties and these additional properties are called direct-properties. A direct-property name has to be unique; it cannot be the same as an inherited-property name. All valid derived-type instances at all times are also valid base-type instances and can be substituted for the parent instance. In the EDM a derived Entity Type always inherits the definition of its key from its base type.

Function Imports are also supported by the EDM. A Function Import is conceptually similar to a method declaration in a header file, in that it defines a function signature, but includes no definition. The parameters and Return Type of the Function Import must either be one of the EDM's built-in primitive types, one of the structural types defined in the rest of the model or collection of these types.

Entity Sets, Association Sets and Function Imports are grouped into one or more Entity Containers. Entity Containers are conceptually similar to databases, however since Entity Types, Association Types and Complex Types are declared outside an EntityContainer this means they can be re-used across Entity Containers.

An example of a Model defined using CSDL is shown in Section 3.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

.NET Framework
XML namespace

The following terms are specific to this document:

alias: A simple identifier that is typically used as a short name for a **namespace**.

alias qualified name: A QualifiedName that is used to refer to a **StructuralType**, except the **namespace** is replaced by the **namespace's alias**. For example if an EntityType called "Person" is defined in the "Model.Business" **namespace**, and that **namespace** has been given the **alias** "Self", then the alias qualified name for the person EntityType is "Self.Person".

annotation: Any custom, application-specific extension to an instance of **CSDL** through the use of custom attributes and elements that are not a part of this **CSDL** specification.

association: A named independent relationship between two EntityType definitions. Associations in the **EDM** are first class concepts and are always bi-directional; indeed the first class nature of associations helps distinguish the Entity Data Model from the Relational Model.

association end: Every **association** includes two association ends, which specify the EntityTypes that are related, the roles of each of those EntityTypes in the **association**, and the **cardinality** rules for each end of the **association**.

cardinality: The measure of the number of elements in a set.

collection: This element is used when declaring a FunctionImport whose parameter or return type is not a single **EDM type** but many. For example, a FunctionImport may return a collection of customers, that is, collection(Model.Customer).

conceptual schema definition language (CSDL): Conceptual schema definition language (CSDL) is a language based on XML that can be used for defining conceptual models based on the Entity Data Model.

CSDL Version 1.0: An older version of **CSDL** that has a slightly reduced set of capabilities, which are called out in this document. Version 1.0 CSDL references this **XML namespace**: http://schemas.microsoft.com/ado/2006/04/edm.

CSDL Version 1.1: An older version of CSDL defined immediately prior to Version 1.2. Version 1.1 CSDL documents references this **XML namespace**: http://schemas.microsoft.com/ado/2007/05/edm.

CSDL Version 1.2: An older version of CSDL defined immediately prior to Version 2.0. Version 1.2 CSDL documents references this **XML namespace**: http://schemas.microsoft.com/ado/2008/01/edm.

CSDL Version 2.0: The version of CSDL that is the focus of this document. Version 2.0 CSDL documents reference this **XML namespace**: http://schemas.microsoft.com/ado/2008/09/edm.

declared property: A property statically declared by a <Property> element as part of the definition of a **StructuralType**. For example, in the context of an EntityType, this property includes all properties of an EntityType represented by the <Property> child elements of the <EntityType> element which defines the EntityType.

derived type: All types with a specified BaseType are said to derive from the BaseType or be derived types of the BaseType. Only ComplexType and EntityType can define a BaseType.

dynamic property: An instance of an OpenEntityType may include additional nullable properties (of type EDMSimpleType or ComplexType) beyond its **declared properties**. The set of additional properties, and the type of each, may vary between instances of the same OpenEntityType. Such additional properties are referred to as dynamic properties and do not have a representation in a CSDL document. If an instance of an OpenEntityType does not include a value for a dynamic property named *N*, then the instance must be treated as if it included *N* with a value of null. A dynamic property of an OpenEntityType must not have the same name as a **declared property** on the same OpenEntityType.

Entity Data Model (EDM): The Entity Data Model (EDM) as described in section 1.0.

EDM type: A categorization that includes all of the following types: EDMSimpleType, ComplexType, EntityType, and **association**.

entity: An instance of an EntityType element that has a unique identity and an independent existence, and is an operational unit of consistency.

facet: This element provides information that specializes the usage of a type – for instance, you can use the precision (that is, accuracy) facet to define the precision of a DateTime **property**.

identifier: A string value that is used to uniquely identify a component of the **CSDL** and is of type SimpleIdentifier.

in scope: This document refers to various XML constructs as being in scope. When something is in scope, it is visible or can be referenced, assuming all other applicable rules are satisfied. Types that are in scope include all EDMSimpleType types and **StructuralType** types that are defined in **namespaces** that are in scope. **Namespaces** that are in scope include the **namespace** of the current **schema** and other **namespaces** referenced in the current **schema** with <Using> elements.

namespace: A name defined on the **schema** that is then subsequently used to prefix **identifiers** to form the **namespace qualified name** of a **StructuralType**. **CSDL** enforces a maximum length of 512 characters for namespace values.

namespace qualified name: A QualifiedName that is used to refer to **StructuralTypes** using the name of the **namespace**, followed by a period, followed by the name of the **StructuralType**.

property: An EntityType can have one or more properties of the specified EDMSimpleType, or ComplexType. A property can be a **declared property** or a **dynamic property**. (In CSDL version 1.2, **dynamic properties** are only defined for use with OpenEntityType instances.)

referential constraint: A constraint on the keys contained in the **association** type. The ReferentialConstraint **CSDL** construct is used for defining referential constraints.

schema: All **EDM types** are contained within some **namespace**. The **schema** concept defines a **namespace** that describes the scope of **EDM types**.

StructuralType: A type that has members that define its structure. ComplexTypes, EntityTypes and Associations are all StructuralTypes.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2   References

1.2.1   Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[ECMA-334] ECMA International, "C# Language Specification", ECMA-334, June 2006, http://www.ecma-international.org/publications/standards/Ecma-334.htm

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

[XML1.0] Bray, T., Paoli, J., Sperberg-McQueen, C.M., and Maler, E., "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C Recommendation, October 2000, http://www.w3.org/TR/2000/REC-xml-20001006

[XMLNS-2ED] World Wide Web Consortium, "Namespaces in XML 1.0 (Second Edition)", August 2006, http://www.w3.org/TR/2006/REC-xml-names-20060816/

[XMLSCHEMA1] Thompson, H.S., Ed., Beech, D., Ed., Maloney, M., Ed., and Mendelsohn, N., Ed., "XML Schema Part 1: Structures", W3C Recommendation, May 2001, http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

1.2.2   Informative References

[MC-EDMX] Microsoft Corporation, "Entity Data Model for Data Services Packaging Format", February 2009.

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[MS-ODATA] Microsoft Corporation, "Open Data Protocol (OData) Specification", January 2011.

1.3   Overview

Conceptual Schema Definition Language (**CSDL**) is an XML-based file format that describes the **Entity Data Model**, and is based on standards defined in [XML1.0] and [XMLSCHEMA1]. The root of the CSDL is a <Schema> element. Below that root, these sub-elements are supported: <Using>, <EntityType>, <ComplexType>, <Association> and <EntityContainer>. In CSDL 2.0 and higher versions, <Schema> elements may have <Function> as a sub-element. <EntityContainers> conceptually represent a <DataSource>, and can contain <EntitySet>, <AssociationSet> and <FunctionImport> sub-elements.

Conceptually a CSDL file has an overall structure that resembles the following:

```
<Schema>
   <Using/>
   <Using/>

   <EntityType/>
   <EntityType/>
   <ComplexType/>

   <Association/>
   <Association/>
   <Function/>
   <Function/>
   <EntityContainer>
      <EntitySet/>
      <EntitySet/>

      <AssociationSet/>
      <AssociationSet/>

      <FunctionImport/>
      <FunctionImport/>
   </EntityContainer>
   <EntityContainer/>
</Schema>
```

Note This is not a detailed specification, which follows. It is only meant to provide a visual overview.

1.4   Relationship to Protocols and Other Structures

Both Entity Data Model for Data Services Packaging Format [MC-EDMX] and Atom Publishing Protocol: Data Services URI and Payload Extensions [MS-ODATA] use the structures defined in CSDL.

1.5   Applicability Statement

CSDL is an XML format that describes the structure and semantics of the Entity Data Model **schemas**. **Identifiers**, such as Names, **Namespaces**, and so on, are all case sensitive.

EDM is a specification for defining conceptual data models. Applications can use the EDM to define a conceptual model that describes the **entity**, relationships, and sets required in the domain served by the application.

1.6   Versioning and Localization

This document describes **CSDL version 1.0**, **CSDL version 1.1**, **CSDL version 1.2**, and **CSDL version 2.0**. Aspects of older CSDL version that do not apply to newer versions are specifically highlighted.

1.7   Vendor-Extensible Fields

CSDL supports application-specific customization and extension through the use of **Annotations**. These Annotations allow applications to embed application-specific or vendor-specific information

into CSDL. The format does not specify how to process these custom-defined structures or how to distinguish structures from multiple vendors or layers. Parsers of the CSDL can ignore Annotations that are not expected or not understood.

Annotations can be of two types: AnnotationAttribute and AnnotationElement.

An AnnotationAttribute is a custom XML attribute applied to a CSDL Element. The attribute can belong to any **XML namespace** (as defined in [XML Namespaces1.0]) that is not in the list of reserved XML namespaces for CSDL. Consult the reference for each CSDL element within this document to determine whether AnnotationAttribute can be used for that element.

The reserved XML namespaces for CSDL are:

http://schemas.microsoft.com/ado/2006/04/edm

http://schemas.microsoft.com/ado/2007/05/edm

http://schemas.microsoft.com/ado/2008/01/edm

http://schemas.microsoft.com/ado/2008/09/edm

2 Structures

2.1 Elements

2.1.1 Schema

The <Schema> is the top-level CSDL construct that allows creation of namespace.

The contents of a namespace can be defined by one or more <Schema> instances. The identifiers used to name types MUST be unique within a **Namespace**. For instance, an **EntityType** cannot have the same name as a **ComplexType** within the same namespace. The **Namespace** forms a part of the type's fully qualified name.

Example:

```
<Schema Alias="Model" Namespace="Test.Simple.Model"
xmlns:edm="http://schemas.microsoft.com/ado/2008/01/edm"
xmlns="http://schemas.microsoft.com/ado/2008/09/edm">
```

The following rules apply to the <Schema> element.

- The CSDL document MUST have the <Schema> element as its root element.

- The **Namespace** attribute MUST be defined for each <Schema> element. **Namespace** is of type **QualifiedName**. A namespace is a logical grouping of **EntityTypes**, **ComplexTypes**, and **Associations**.

- A schema **Namespace** MUST NOT be "System", "Transient" or "Edm".

- A schema definition can span across more than one CSDL document.

- **Alias** attribute can be defined on <Schema> element. **Alias** is of the type **SimpleIdentifier**.

- <Schema> can contain any number of AnnotationAttributes. The full names of AnnotationAttributes MUST NOT collide.

- <Schema> can contain 0 or more of the following subelements. The elements can appear in any given order.

  - <Using>

  - <Association>

  - <ComplexType>

  - <EntityType>

  - <EntityContainer>

- In CSDL 2.0 and higher versions, <Schema> can contain 0 or more of the following sub-element.

  - <Function>

- <Schema> can contain any number of AnnotationElement elements.

- AnnotationElement elements MUST only appear after all other **Schema** sub-elements.

Element	Schema			
Attributes	Name		Required	
	Namespace		Yes	
	Alias		No	
	AnnotationAttribute		No	
Sub elements	Name		Occurrence	
			Min	Max
	Choice	Using	0	Unbounded
		Association	0	Unbounded
		ComplexType	0	Unbounded
		EntityType	0	Unbounded
		EntityContainer	0	Unbounded
	AnnotationElement		0	Unbounded

All sub-elements MUST appear in the order indicated. For all sub-elements within a given Choice, the sub-elements can be ordered arbitrarily.

2.1.2 EntityType

An entity is an instance of an <EntityType>. It has a unique identity, independent existence and forms the operational unit of consistency. Intuitively, <EntityTypes> model the top-level concepts within a data model - such as Customers, Orders, Suppliers, and so on (to take the example of a typical line-of-business system). An entity instance represents one particular instance of the <EntityType> such as a specific customer or a specific order. An <EntityType> can be either abstract or concrete. An abstract <EntityType> cannot be instantiated.

An <EntityType> has a **Name**, a payload consisting of one or more **declared properties**, and a <Key> that describes the set of **properties** whose values uniquely identify an entity within an entity set.

In CSDL 1.2 and higher versions, an <EntityType> can be an OpenEntityType, denoted by the presence of an **OpenType**="true" attribute; if so, then the set of properties associated with the <EntityType> can, in addition to declared properties, include **dynamic properties**.

The type of a <Property> can be an **EDMSimpleType** or **ComplexType**.

Example

```
<EntityType Name="Customer">
  <Key>
    <PropertyRef Name="CustomerId" />
  </Key>
  <Property Name="CustomerId" Type="Int32" Nullable="false" />
  <Property Name="FirstName" Type="String" Nullable="true" />
```

```
        <Property Name="LastName" Type="String" Nullable="true" />
        <Property Name="AccountNumber" Type="Int32" Nullable="true" />
        <NavigationProperty Name="Orders" Relationship="Model1.CustomerOrder" FromRole="Customer"
  ToRole="Order" />
    </EntityType>
```

The following rules apply to the <EntityType> element.

- <EntityType> MUST have a Name attribute defined. Name attribute is of type SimpleIdentifier. Name attribute represents the name of this <EntityType>.

- The **Name** of an <EntityType> MUST be unique across all <EntityType> types, **Association** types, and ComplexType types defined in the same namespace.

- <EntityType> can derive from a **BaseType**, which is used to specify the parent type of a **derived type**. The derived type inherits properties from the parent type.

- If a **BaseType** is defined, it MUST be a **Namespace Qualified Name** or an **Alias Qualified Name** of an <EntityType> that is in scope.

- An <EntityType> MUST NOT introduce an inheritance cycle via the **BaseType** attribute.

- An <EntityType> can have its **Abstract** attribute set to true. By default, the **Abstract** attribute is false.

- An <EntityType> can contain any number of AnnotationAttribute attributes. The full name of AnnotationAttribute MUST NOT collide.

- An <EntityType> element can contain at most one <Documentation> element.

- An <EntityType> MUST either define a <Key> or derive from a **BaseType**. Derived <EntityTypes> MUST NOT define a <Key>. A key forms the identity of the <Entity>.

- An <EntityType> can have any number of <Property> and <NavigationProperty> elements in any given order.

- <EntityType> <Property> sub-elements MUST be uniquely named within the inheritance hierarchy for the <EntityType>. <Property> sub-elements and <NavigationProperty> sub-elements MUST NOT have the same name as their declaring <EntityType>.

- An <EntityType> can contain any number of AnnotationElement element block.

- In CSDL 1.2 and higher versions, an <EntityType> representing an OpenEntityType MUST have an **OpenType** attribute defined with its value equal to "true".

- In CSDL 1.2 and higher versions, an <EntityType> which derives from an OpenEntityType is itself an OpenEntityType. Such a derived <EntityType> MUST NOT have an **OpenType** attribute with its value equal to "false", but can have an **OpenType** attribute defined with its value equal to "true".

Element	EntityType		
Attributes	Name	Required	
	Name	Yes	
	BaseType	No	
	Abstract	No (default=FALSE)	
	AnnotationAttribute	No	
	OpenType	No	
Sub elements	Name	Occurrence	
		Min	Max
	Documentation	0	1
	Key	0	1
	Property (Choice)	0	Unbounded
	NavigationProperty (Choice)	0	Unbounded
	AnnotationElement	0	Unbounded

All sub-elements MUST appear in the order indicated. For all sub-elements within a given Choice, the sub-elements can be ordered arbitrarily.

2.1.3 Property

The declared properties of an <EntityType> or <ComplexType> are defined using the <Property> element. <EntityType> and <ComplexType> can have <Property> elements. <Property> can be an **EDMSimpleType** or <ComplexType>. A declared property description consists of its name, type and a set of facets, such as Nullable or Default. **Facets** describe further behavior of a given type; they are optional to define.

Example:

```
<Property Name="ProductName" Type="String" Nullable="false" MaxLength="40">
```

The following rules apply to the <Property> element.

- <Property> MUST define Name attribute.

- <Property> MUST have the Type defined.

- <Property> Type MUST be either an EDMSimpleType or a Namespace Qualified Name or Alias Qualified Name of a ComplexType that is in scope.

- <Property> can define a Nullable facet. The default value is Nullable=true. (Any Property that has a Type of ComplexType, MUST also define a Nullable attribute which MUST be set to false.)

- The following facets are optional to define on <Property>:

  - Default

- MaxLength

- FixedLength

- Precision

- Scale

- Unicode

- Collation

- In CSDL 1.1 and higher versions, a <Property> element can define **CollectionKind**. The possible values are "None", "List" and "Bag".

- <Property> can define **ConcurrencyMode**. The possible values are "None" and "Fixed". However, for an <EntityType> that has a corresponding <EntitySet> defined, any <EntityTypes> derived from it MUST NOT define any new <Property> with **ConcurrencyMode** set to a value other than "None".

- <Property> can contain any number of AnnotationAttribute attributes. The full names of these AnnotationAttributes MUST NOT collide.

- <Property> element can contain maximum of one <Documentation> element.

- <Property> can contain any number of AnnotationElements.

- Sub-elements for <Property> MUST appear in this sequence: <Documentation>, AnnotationElements.

Element	Property	
Attributes	Name	Required
	Name	Yes
	Type	Yes
	Nullable	No (default=TRUE)
	DefaultValue	No
	MaxLength	No
	FixedLength	No
	Precision	No
	Scale	No
	Unicode	No
	Collation	No
	ConcurrencyMode	No
	AnnotationAttribute	No

Sub elements	Name	Occurrence	
		Min	Max
	Documentation	0	1
	AnnotationElement	0	Unbounded

All sub-elements MUST appear in the order indicated.

2.1.4 NavigationProperty

<NavigationProperty> elements define non-structural properties on entities that allow for navigation from one <Entity> to another via a relationship. Standard properties describe a value associated with an entity, while navigation properties describe a navigation path over a relationship. For example, given a relationship between Customer and Order entities, an Order <EntityType> may describe a <NavigationProperty> "OrderedBy" that represents the Customer instance associated with that particular Order instance.

Example:

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

The following rules apply to the <NavigationProperty> element.

- <NavigationProperty> MUST have a **Name** defined.

- <NavigationProperty> MUST have a **Relationship** attribute defined.

- Relationship MUST be either a Namespace Qualified Name or an Alias Qualified Name of an <Association> that is in scope.

- <NavigationProperty> MUST have a **ToRole** defined. **ToRole** specifies the other end of the relationship. **ToRole** MUST refer to one of the Role names defined on the <Association>.

- <NavigationProperty> MUST have a **FromRole** defined. **FromRole** is used to establish the starting point for the navigation and MUST refer to one of the Role names defined on the <Association>.

- <NavigationProperty> can contain any number of AnnotationAttribute attributes. The full names of AnnotationAttribute MUST NOT collide.

- <NavigationProperty> element can contain a maximum of one <Documentation> element.

- <NavigationProperty> can contain any number of AnnotationElement elements.

- Sub-elements for <NavigationProperty> MUST appear in this sequence: <Documentation>, AnnotationElement.

Element	NavigationProperty		
Attributes	Name	Required	
	Name	Yes	
	Relationship	Yes	
	ToRole	Yes	
	FromRole	Yes	
	AnnotationAttribute	No	
Sub elements	Name	Occurrence	
		Min	Max
	Documentation	0	1
	AnnotationElement	0	Unbounded

All sub-elements MUST appear in the order indicated.

2.1.5 Entity Key

<Key> describes which <Property> elements form a key that can uniquely identify instances of an <EntityType>. Any set of non-nullable, immutable, <EDMSimpleType> declared properties can serve as the key.

Example:

```
<Key>
  <PropertyRef Name="CustomerId" />
</Key>
```

The following rules apply to the <Key> element.

- <Key> can contain any number of AnnotationAttribute attributes. The full names of AnnotationAttributes MUST NOT collide.

- <Key> MUST have one or more <PropertyRef> sub-elements.

- In 2.0 and higher versions, <Key> can contain any number of AnnotationElement elements.

Element	EntityKey		
Attributes	Name	Required	
	AnnotationAttribute	No	
Sub elements	Name	Occurrence	
		Min	Max
	PropertyRef	1	Unbounded
	AnnotationElement	0	Unbounded

All sub-elements MUST appear in the order indicated.

2.1.6 PropertyRef

<PropertyRef> element refers to a declared property of an EntityType.

Example:

```
<PropertyRef Name="CustomerId" />
```

The following rules apply to the <PropertyRef> element.

- <PropertyRef> can contain any number of AnnotationAttribute attributes. The full names of AnnotationAttributes MUST NOT collide.

- <PropertyRef> MUST define the name attribute. Name attribute refers to the name of a <Property> defined in the declaring <EntityType>.

- In 2.0 and higher versions, <PropertyRef> can contain any number of AnnotationElement elements.

Element	PropertyRef		
Attributes	Name	Required	
	Name	Yes	
	AnnotationAttribute	No	
Sub elements	Name	Occurrence	
		Min	Max
	AnnotationElement	0	Unbounded

All sub-elements MUST appear in the order indicated.

2.1.7 ComplexType

A <ComplexType> element represents a set of related information. Like an <EntityType> element, a <ComplexType> element consists of one or more properties of EDMSimpleType or complex type. However, unlike an <EntityType> element, a <ComplexType> element cannot have a <Key> element or any <NavigationProperty> elements.

A <ComplexType> element provides a mechanism to create declared properties with a rich (structured) payload. Its definition includes its name and payload. The payload of a <ComplexType> is very similar to that of an <EntityType>.

Example:

```
<ComplexType Name="CAddress">
  <Documentation>
      <Summary>This complextype describes the concept of an Address</Summary>
      <LongDescription>This complextype describes the concept of an Address for use with
Customer and other Entities</LongDescription>
  </Documentation>
  <Property Name="StreetAddress" Type="String">
      <Documentation>
          <LongDescription>StreetAddress contains the string describing the address of the
street associated with an address</LongDescription>
      </Documentation>
  </Property>
  <Property Name="City" Type="String" />
  <Property Name="Region" Type="String" />
  <Property Name="PostalCode" Type="String" />
</ComplexType>
```

The following rules apply to the ComplexType element.

- <ComplexType> MUST have a **Name** attribute defined. **Name** attribute is of type **SimpleIdentifier**. <Name> attribute represents the name of this <ComplexType>.

- <ComplexType> **Name** MUST be unique across all <EntityType> types, <Association> types and <ComplexType> types defined in the same namespace.

- In CSDL 1.1 and higher versions, a <ComplexType> can derive from a **BaseType**. **BaseType** MUST be either the **NamespaceQualifiedName** or **AliasQualifiedName** of another <ComplexType> that is **in scope**.

- A <ComplexType> MUST NOT introduce an inheritance cycle via the **BaseType** attribute.

- In CSDL 1.1 and higher versions, <ComplexType> can have its **Abstract** attribute set to true. By default, **Abstract** is false.

- <ComplexType> can contain any number of AnnotationAttribute attributes. The full name of the AnnotationAttributes MUST NOT collide.

- A <ComplexType> element can contain a maximum of one <Documentation> element.

- <ComplexType> can have any number of <Property> elements.

- In CSDL 1.1 and higher versions, (CSDL version 1.0 does not support inheritance) the property names of a <ComplexType> MUST be uniquely named within the inheritance hierarchy for the

<ComplexType>. <ComplexType> properties MUST NOT have the same name as their declaring <ComplexType> or any of its base types.

- <ComplexType> can contain any number of AnnotationElement elements.

- Subelements for <ComplexType> MUST appear in this sequence: <Documentation>, <Property>, AnnotationElements.

Element	ComplexType		
Attributes	Name	Required	
	Name	Yes	
	BaseType	No	
	Abstract	No	
	AnnotationAttribute	No	
Sub elements	Name	Occurrence	
		Min	Max
	Documentation	0	1
	Property	0	Unbounded
	AnnotationElement	0	Unbounded

All sub-elements MUST appear in the order indicated.

2.1.8 Association

<Association> defines a peer-to-peer relationship between participating <EntityTypes> and can support different multiplicities at the two ends. OnDelete operational behavior can be specified at any end of the relationship.

A classic example of an association is the relationship between the Customer and Order entities. Typically, this relationship has the following characteristics:

- Multiplicity: each Order is associated with exactly one Customer. Every Customer has zero or more Orders.

- Operational Behavior: OnDelete Cascade; when an Order with one or more OrderLines is deleted, the corresponding OrderLines also get deleted.

Example:

```
<Association Name="CustomerOrder">
  <End Type="Model1.Customer" Role="Customer" Multiplicity="1" />
  <End Type="Model1.Order" Role="Order" Multiplicity="*" />
</Association>
```

The following rules apply to the <Association> element.

- <Association> MUST have a **Name** attribute defined. **Name** attribute is of type **SimpleIdentifier**.

- An <Association> **Name** MUST be unique across all <EntityTypes>, <Associations> and <ComplexTypes> defined in the same namespace.

- <Association> can contain any number of AnnotationAttributes. The full names of the AnnotationAttributes MUST NOT collide.

- An <Association> element can contain a maximum of one <Documentation> element.

- <Association> MUST have exactly two <End> elements defined.

- <Association> can have one <ReferentialConstraint> defined.

- <Association> can contain any number of AnnotationElements.

- Sub-elements for <Association> MUST appear in this sequence: <Documentation>, <AssociationEnds>, <ReferentialConstraint>, AnnotationElements.

Element	Association		
Attributes	Name	Required	
	Name	Yes	
	AnnotationAttribute	No	
Sub elements	Name	Occurrence	
		Min	Max
	Documentation	0	1
	End	2	2
	ReferentialConstraint	0	1
	AnnotationElement	0	Unbounded

All sub-elements MUST appear in the order indicated.

2.1.9 Association End

For a given <Association>, <End> defines one side of the relationship. <End> defines what type is participating in the relationship, multiplicity or the **cardinality**, and if there are any operation associations, like cascade delete.

Example:

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

The following rules apply to the <Association> <End> element.

- <End> MUST define the <EntityType> for this end of the relationship.

- <EntityType> MUST be either a Namespace Qualified Name or an Alias Qualified Name of an <EntityType> that is in scope.

- <End> MUST specify the **Multiplicity** of this end.

- <End> can specify the Role name.

- <End> can contain any number of AnnotationAttributes. The full names of AnnotationAttribute MUST NOT collide.

- <End> can contain a maximum of one <Documentation> element.

- At most, one <OnDelete> operation can be defined on a given <End>.

- <End> can contain any number of AnnotationElements.

- Sub-elements for <End> MUST appear in this sequence: <Documentation>, <OnDelete>, AnnotationElements.

Element	Association End		
Attributes	Name	Required	
	Type	Yes	
	Role	No	
	Multiplicity	Yes	
	AnnotationAttribute	No	
Sub elements	Name	Occurrence	
		Min	Max
	Documentation	0	1
	OnDelete	0	1
	AnnotationElement	0	Unbounded

All sub-elements MUST appear in the order indicated.

2.1.10   OnDelete

<OnDelete> is a trigger that is associated with a relationship. The action is performed on one end of the relationship when the state of the other side of the relationship changes.

Example:

```
<Association Name="CProductCategory">
  <End Type="Self.CProduct" Multiplicity="*" />
  <End Type="Self.CCategory" Multiplicity="0..1">
    <OnDelete Action="Cascade" />
  </End>
```

The following rules apply to the OnAction element.

- <OnDelete> MUST specify the **Action**.

- <OnDelete> can contain any number of AnnotationAttributes. The full names of the AnnotationAttributes MUST NOT collide.

- <OnDelete> element can contain a maximum of one <Documentation> element.

- <OnDelete> can contain any number of AnnotationElement elements.

- Sub-elements for <OnDelete> MUST appear in this sequence: <Documentation>, AnnotationElement.

Element	OnDelete		
Attributes	Name	Required	
	Action	Yes	
	AnnotationAttribute	No	
Sub elements	Name	Occurrence	
		Min	Max
	Documentation	0	1
	AnnotationElement	0	Unbounded

All sub-elements MUST appear in the order indicated.

2.1.11 ReferentialConstraint

In EDM, <ReferentialConstraints> can exist between the key of one entity type and primitive property (or properties) of another associated entity type (In 1.2 and earlier versions the <ReferentialConstraint> can only exist between the key properties of associated entities). The two entity types are in a Principal-to-Dependent relationship, which can also be thought of as a type of parent-child relationship. When entities are related by an <Association> that specifies a <ReferentialConstraint> between the keys of the two entities, then the dependent entity object cannot exist without a valid relationship to a parent entity object.

The <ReferentialConstraint> MUST specify which of the ends is the <Principal> Role and which of the ends is the <Dependent> Role for the <ReferentialConstraint>.

Example:

```
<Association Name="FK_Employee_Employee_ManagerID">
  <End Role="Employee" Type="Adventureworks.Store.Employee" Multiplicity="1" />
  <End Role="Manager" Type="Adventureworks.Store.Manager" Multiplicity="0..1" />
  <ReferentialConstraint>
    <Principal Role="Employee">
      <PropertyRef Name="EmployeeID" />
    </Principal>
    <Dependent Role="Manager">
      <PropertyRef Name="ManagerID" />
    </Dependent>
  </ReferentialConstraint>
</Association>
```

The following rules apply to the <ReferentialConstraint> element.

- <ReferentialConstraint> MUST define exactly one <Principal> end element and exactly one <Dependent> end element.

- <ReferentialConstraint> can contain any number of AnnotationAttribute attributes. The full name AnnotationAttribute MUST NOT collide.

- A <ReferentialConstraint> element can contain maximum of one <Documentation> element.

- <ReferentialConstraint> can contain any number of AnnotationElement elements.

- Subelements for <ReferentialConstraint> MUST appear in this sequence: <Documentation>, <Principal>, <Dependent>, AnnotationElement.

Element	ReferentialConstraint			
Attributes	Name		Required	
	AnnotationAttribute		No	
Sub elements	Name		Occurrence	
			Min	Max
	Documentation		0	1
	Principal		1	1
	Dependent		1	1
	AnnotationElement		0	Unbounded

All sub-elements MUST appear in the order indicated.

2.1.12 ReferentialConstraint Role

When defining <ReferentialConstraints>, Role MUST be used to indicate which end of the association is the Principal and which end of the relationship is the Dependent. Thus, the <ReferentialConstraint> MUST contain two Role definitions: the <Principal> and <Dependent>.

<ReferentialConstraint> Role usage MUST also conform to the ordering rules for the sub-elements of <ReferentialConstraint> as defined in the <ReferentialConstraint> section (2.1.11).

An example of the <ReferentialConstraint> role is shown below, which defines <Principal> and <Dependent> elements.

Example:

```
<ReferentialConstraint>
  <Principal Role="Employee">
    <PropertyRef Name="EmployeeID" />
  </Principal>
  <Dependent Role="Manager">
    <PropertyRef Name="ManagerID" />
  </Dependent>
</ReferentialConstraint>
```

2.1.12.1 Principal

The following example shows the usage of the <Principal> role in defining a
<ReferentialConstraint>.

Example:

```
<Principal Role="Employee">
  <PropertyRef Name="EmployeeID" />
</Principal>
```

The following rules apply to the <Principal> role element.

- One <Principal> role MUST be used to define the <Principal> end of the <ReferentialConstraint>.

- Each <Principal> role MUST specify one and only one <Role> attribute. <Role> is of type **SimpleIdentifier**.

- <Principal> MUST have one or more <PropertyRef> elements. Each <PropertyRef> element MUST specify a name using the **Name** attribute.

- For each <Principal>, a <PropertyRef> definition MUST NOT have the same **Name** value specified as another <PropertyRef>.

- <PropertyRef> MUST be used to specify the properties that participate in the <Principal> role of the <ReferentialConstraint>.

- Each <PropertyRef> element on the <Principal> MUST correspond to a <PropertyRef> on the <Dependent>. The <Principal> and the <Dependent> of the <ReferentialConstraint> MUST contain the same number of <PropertyRef> elements. The <PropertyRef> elements on the <Dependent> MUST be listed in the same order as the corresponding <PropertyRef> elements on the <Principal>.

- The <Principal> of a <ReferentialConstraint> MUST specify all properties constituting the <Key> of the <EntityType> that forms the <Principal> of the <ReferentialConstraint>.

- The Multiplicity of the <Principal> role MUST be 1. For CSDL 2.0 and higher versions, the Multiplicity of the <Principal> <Role> can be 1 or 0..1.

- The datatype of each property defined in the <Principal> role MUST be the same as the datatype of the corresponding property specified in the <Dependent> role.

- In 2.0 and higher versions, <Principal> can contain any number of AnnotationElement elements.

- Sub-elements for <Principal> MUST appear in this sequence: <PropertyRef>, AnnotationElement.

Element	ReferentialConstraintRoleElement		
Attributes	Name	Required	
	Role	Yes	
	AnnotationAttribute	No	
Sub elements	Name	Occurrence	
		Min	Max
	PropertyRef	1	Unbounded
	AnnotationElement	0	Unbounded

All sub-elements MUST appear in the order indicated.

2.1.12.2   Dependent

The following example shows the usage of the <Dependent> role element in defining a <ReferentialConstraint>.

Example:

```
<Dependent Role="Manager">
  <PropertyRef Name="ManagerID" />
</Dependent>
```

The following rules apply to the <Dependent> role:

- One <Dependent> role MUST be used to define the <Dependent> end of the <ReferentialConstraint>. This element name MUST be <Dependent>.

- Each <Dependent> role MUST specify one and only one <Role> attribute. <Role> is of type **SimpleIdentifier**.

- <Dependent> MUST have one or more <PropertyRef> elements. Each <PropertyRef> element MUST specify a name using the **Name** attribute.

- For each <Dependent>, a <PropertyRef> definition MUST NOT have the same **Name** value specified as another <PropertyRef>.

- <PropertyRef> MUST be used to specify the properties that participate in the <Dependent> role of the <ReferentialConstraint>.

- Each <PropertyRef> element on the <Principal> MUST correspond to a <PropertyRef> on the <Dependent>. The <Principal> and the <Dependent> of the <ReferentialConstraint> MUST contain the same number of <PropertyRef> elements. The <PropertyRef> elements on the <Dependent> MUST be listed in the same order as the corresponding <PropertyRef> elements on the <Principal>.

- The data type of each property defined in the <Principal> <Role> MUST be the same as the data type of the corresponding property specified in the <Dependent> <Role>.

- In 2.0 and higher versions, <Dependent> can contain any number of AnnotationElement elements.

- Subelements for <Dependent> MUST appear in this sequence: <PropertyRef>, AnnotationElement.

Element	ReferentialConstraintRoleElement		
Attributes	Name	Required	
	Role	Yes	
	AnnotationAttribute	No	
Sub elements	Name	Occurrence	
		Min	Max
	PropertyRef	1	Unbounded
	AnnotationElement	0	Unbounded

All subelements MUST appear in the order indicated.

2.1.13 Using

<Using> imports the contents of the specified namespace. A schema can refer to contents of another schema or namespace by importing it with the <Using> clause. The imported namespace can be associated with an **alias** which is then used to refer to its types, or the types can be directly used by specifying its fully qualified name.

Note Semantically, <Using> is closer to a merge; unfortunately, the name does not reflect this. If it was truly "using", structures in the schema being used would be unaffected. However, because a dependent schema can derive an <EntityType> from an <EntityType> declared in the original schema, this has the potential side-effect of changing what might be found in <EntitySets> declared in the schema being used.

Example:

```
<Using Namespace="Microsoft.Samples.Northwind.Types"  Alias="Types" />
```

The following rules apply to the <Using> element.

- <Using> MUST have a **Namespace** attribute defined. **Namespace** is of type **QualifiedName**.

- <Using> MUST have an **Alias** attribute defined. Alias is of type **SimpleIdentifier**. The **Alias** can be used as a short hand for referring to the **Namespace** linked to that alias via the <Using> element.

- <Using> can contain any number of AnnotationAttributes. The full names of the AnnotationAttributes MUST NOT collide.

- <Using> can contain a maximum of one <Documentation> element.

- <Using> can contain any number of AnnotationElement elements.

- If a <Documentation> element is defined, it MUST come before any AnnotationElements.

Element	Using	
Attributes	Name	Required
	Namespace	Yes
	Alias	Yes
	AnnotationAttribute	No
Sub elements	Name	Occurrence
		Min	Max
	Documentation	0	1
	AnnotationElement	0	Unbounded

All sub-elements MUST appear in the order indicated.

2.1.14   EntityContainer

<EntityContainer> is conceptually similar to a database or data source. It groups <EntitySet>, <AssociationSet> and <FunctionImport> sub-elements that represent a data source.

Example:

```
<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>
```

The following rules apply to the <EntityContainer> element.

- <EntityContainer> MUST have a **Name** attribute defined. The **Name** attribute is of type **SimpleIdentifier**.

- <EntityContainer> can define an Extends attribute, which MUST, if present, refer to another <EntityContainer> in scope by name.

- <EntityContainers> that extend another <EntityContainer> inherit all the extended <EntitySet>, <AssociationSet> and <FunctionImport> sub-elements from that <EntityContainer>.

- <EntityContainer> can contain maximum of one <Documentation> element.

- <EntityContainer> can contain any number of AnnotationAttributes. The full name of these AnnotationAttributes MUST NOT collide.

- <EntityContainer> can contain any number of <FunctionImport>, <EntitySet> and <AssociationSet> elements, which can be defined in any order.

- <FunctionImport>, <EntitySet> and <AssociationSet> names within an <EntityContainer> MUST NOT collide.

- If present, the <Documentation> sub-element MUST precede <FunctionImport>, <EntitySet> and <AssociationSet> sub-elements.

- In 2.0 and higher versions, <EntityContainer> can contain any number of AnnotationElement elements.

- In the sequence of sub-elements under <EntityContainer>, AnnotationElement MUST follow all other elements.

Element	EntityContainer		
Attributes	Name	Required	
	Name	Yes	
	Extends	No	
	AnnotationAttribute	No	
Sub elements	Name	Occurrence	
		Min	Max
	Documentation	0	1
	FunctionImport	0	Unbounded
	EntitySet	0	Unbounded
	AssociationSet	0	Unbounded
	AnnotationElement	0	Unbounded

(Choice groups FunctionImport, EntitySet, AssociationSet)

All sub-elements MUST appear in the order indicated. For all sub-elements within a given Choice, the sub-elements can be ordered arbitrarily.

2.1.15 FunctionImport

<FunctionImport> element is used to import Stored Procedures or Functions defined in Store Schema Model into Entity Data Model.

Example:

```
<FunctionImport Name="annualCustomerSales" EntitySet="result_annualCustomerSalesSet"
ReturnType="Collection(Self.result_annualCustomerSales)">
  <Parameter Name="fiscalyear" Mode="In" Type="String" />
</FunctionImport>
```

The following rules apply to the <FunctionImport> element.

- <FunctionImport> MUST have a Name attribute defined. Name attribute is of type **SimpleIdentifier**.

- <FunctionImport> can define a **ReturnType**.

- If defined, the **Type** of **ReturnType** MUST be an **EDMSimpleType**, **EntityType** or **ComplexType** that is in scope or a **Collection** of one of these in scope types. (In CSDL version 1.0 the **ReturnType** MUST be a Collection of either **EDMSimpleType** or **EntityType**).

- Types in scope for a <FunctionImport> include: All **EDMSimpleTypes**, and all **EntityTypes** and **ComplexTypes** that are defined in the declaring <Schema> **Namespace**, or schemas that are in scope of the declaring <Schema>.

- If the return type of <FunctionImport> is a **Collection** of Entities, then **EntitySet** attribute MUST be defined.

- If the return type of <FunctionImport> is **ComplexType** or **EDMSimpleType** then **EntitySet** attribute MUST NOT be defined.

- <FunctionImport> can contain any number of AnnotationAttributes. The full names of AnnotationAttributes MUST NOT collide.

- <FunctionImport> element can contain maximum of one <Documentation> element.

- <FunctionImport> can have any number of <Parameter> elements.

- <Parameter> element names inside a <FunctionImport> MUST NOT collide.

- In 2.0 and higher versions, <FunctionImport> can contain any number of AnnotationElement elements.

- Sub-elements for <FunctionImport> MUST appear in this sequence: <Documentation> (if present), <Parameter>, AnnotationElement.

Element	FunctionImport		
Attributes	Name	Required	
	Name	Yes	
	ReturnType	No	
	EntitySet	No	
	MethodAccess	No	
	AnnotationAttribute	No	
Sub elements	Name	Occurrence	
		Min	Max
	Documentation	0	1
	Parameter	0	Unbounded
	AnnotationElement	0	Unbounded

All sub-elements MUST appear in the order indicated.

2.1.16   FunctionImport Parameter

Functions defined in CSDL optionally accept both in and out <Parameters>. Each <Parameter> element MUST have an associated **Name** and **Type** defined.

Example:

```
<FunctionImport Name="GetScalar" ReturnType="Collection(String)">
  <Parameter Name="count" Type="Int32" Mode="Out" />
  <ValueFunctionImport Anything="bogus1" xmlns="FunctionImportAnnotation"/>
</FunctionImport>
```

- The following rules apply to the FunctionImport Parameter element.

- <Parameter> MUST have a **Name** defined.

- **Type** of the <Parameter> MUST be defined. **Type** MUST be an **EDMSimpleType** or **ComplexType**.

- <Parameter> can define the **Mode** of the parameter. Possible values are "In", "Out", and "InOut".

- For a given <Parameters>, **MaxLength** value can be specified.

- **Precision** can be specified for a given <Parameter>.

- **Scale** can be specified for a given <Parameter>.

- <Parameter> can contain any number of AnnotationAttributes. The full name of AnnotationAttribute MUST NOT collide.

- <Parameter> can contain a maximum of one <Documentation> element.

- <Parameter> can contain any number of AnnotationElements.

- Sub-elements for <Parameter> MUST appear in this sequence: <Documentation>, AnnotationElement.

Element	Parameter		
Attributes	Name	Required	
	Name	Yes	
	Type	Yes	
	Mode	No	
	MaxLength	No	
	Precision	No	
	Scale	No	
	AnnotationAttribute	No	
Sub elements	Name	Occurrence	
		Min	Max
	Documentation	0	1
	AnnotationElement	0	Unbounded

All sub-elements MUST appear in the order indicated.

2.1.17 EntitySet

An <EntitySet> is a named set that can contain instances of a specified EntityType and any of the specified **EntityType** subtypes. More than one <EntitySet> for a particular **EntityType** can be defined.

Example:

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

The following rules apply to the <EntitySet> element.

- <EntitySet> MUST have a <Name> attribute defined. <Name> attribute is of type **SimpleIdentifier**.

- <EntitySet> MUST have an **EntityType** defined.

- The **EntityType** of an <EntitySet> MUST be in scope of the <Schema> which declares the <EntityContainer> in which this <EntitySet> resides.

- <EntitySet> can have an abstract **EntityType**. An <EntitySet> for a given **EntityType** can contain instances of that **EntityType** and any of its subtypes.

- Multiple <EntitySets> can be defined for a given **EntityType**.

- <EntitySet> can contain any number of AnnotationAttributes. The full names of AnnotationAttributes MUST NOT collide.

- <EntitySet> elements can contain a maximum of one <Documentation> element.

- <EntitySet> can contain any number of AnnotationElements.

- Sub-elements for <EntitySet> MUST appear in this sequence: <Documentation>, AnnotationElement.

Element	EntitySet		
Attributes	Name	Required	
	Name	Yes	
	EntityType	Yes	
	AnnotationAttribute	No	
Sub elements	Name	Occurrence	
		Min	Max
	Documentation	0	1
	AnnotationElement	0	Unbounded

All sub-elements MUST appear in the order indicated.

2.1.18 AssociationSet

An <AssociationSet> contains relationship instances of the specified **Association**. The **Association** specifies the **EntityTypes** of the two end points while <AssociationSet> specifies the <EntitySet> that corresponds to either these **EntityTypes** directly, or to derived **EntityTypes**. The association-instances contained in the <AssociationSet> relate entity instances belonging to these **EntityTypes**.

Example:

```
<AssociationSet Name="CustomerOrder" Association="Model1.CustomerOrder">
  <End Role="Customer" EntitySet="CustomerSet" />
  <End Role="Order" EntitySet="OrderSet" />
</AssociationSet>
```

The following rules apply to the <AssociationSet> element.

- <AssociationSet> MUST have a **Name** attribute defined. **Name** attributes are of type **SimpleIdentifier**.

- <AssociationSet> MUST have an **Association** attribute defined. The **Association** attribute should specify a Namespace Qualified Name or an Alias Qualified Name of the **Association** that the <AssociationSet> is being defined for.

- The **Association** of an <AssociationSet> MUST be in scope of the <Schema> which declares the <EntityContainer> in which this <AssociationSet> resides.

- <AssociationSet> can contain any number of AnnotationAttributes. The full name of AnnotationAttributes MUST NOT collide.

- <AssociationSet> element can contain a maximum of one <Documentation> element.

- <AssociationSet> MUST have EXACTLY two <Ends> defined.

- <AssociationSet> can contain any number of AnnotationElements.

- Sub-elements for <AssociationSet> MUST appear in this sequence: <Documentation>, <End>, AnnotationElement.

Element	AssociationSet		
Attributes	Name	Required	
	Name	Yes	
	Association	Yes	
	AnnotationAttribute	No	
Sub elements	Name	Occurrence	
		Min	Max
	Documentation	0	1
	End	2	2
	AnnotationElement	0	Unbounded

All sub-elements MUST appear in the order indicated.

2.1.19 AssociationSet End

The <End> element defines the two sides of the <AssociationSet>. This association is defined between the two **EntitySets** declared in an **EntitySet** attribute.

Example:

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

The following rules apply to <End> elements inside an <AssociationSet>.

- <End> element can have the **Role** attribute specified. All <End> elements MUST have the **EntitySet** attribute specified.

- The **EntitySet** MUST be the **Name** of an **EntitySet** defined inside the same <EntityContainer>.

- The <End> element's **Role** MUST map to a **Role** declared on one of the <Ends> of the <Assocation> referenced by the <End> element's declaring <AssociationSet>.

- Each <End>declared by an <AssociationSet> MUST refer to a different **Role**.

- The **EntitySet** for a particular <AssociationSet> <End>, MUST contain either the same **EntityType** as the related <End> **EntityType** on the <Assocation> element, or an **EntityType** derived from that **EntityType**. An <End> element can contain a maximum of one <Documentation> element.

- <End> can contain any number of AnnotationElements.

- Sub-elements for <End> MUST appear in this sequence: <Documentation>, AnnotationElement.

Element	End			
Attributes	Name		Required	
	Role		No	
	EntitySet		Yes	
	AnnotationAttribute		No	
Sub elements	Name		Occurrence	
			Min	Max
	Documentation		0	1
	AnnotationElement		0	Unbounded

All sub-elements MUST appear in the order indicated.

2.1.20 Documentation

The <Documentation> element is used to provide documentation of comments on the contents of the CSDL file.

Example 1: Documentation on <EntityContainer>

```
<EntityContainer Name="TwoThreeContainer">
   <Documentation>
      <Summary>Summary: Entity Container for storing Northwind instances</Summary>
      <LongDescription>LongDescription: This Entity Container is for storing Northwind
instances</LongDescription>
   </Documentation>
   <EntitySet Name="Products" EntityType="Self.Product" />
</EntityContainer>
```

Example 2: Documentation on <EntitySet>

```
<EntitySet Name="Products" EntityType="Self.Product">
   <Documentation>
      <Summary>EntitySet Products is for storing instances of EntityType Product</Summary>
      <LongDescription>This EntitySet having name Products is for storing instances of
EntityType Product</LongDescription>
   </Documentation>
</EntitySet>
```

Example 3: Documentation on <AssociationSet> and <End> role

```
<AssociationSet Name="CategoryProducts" Association="Self.CategoryProduct">
     <Documentation>
         <Summary>AssociationSet CategoryProducts is for storing instances of Association
CategoryProduct</Summary>
         <LongDescription>This AssociationSet having name=CategoryProducts is for storing
instances of Association CategoryProduct</LongDescription>
     </Documentation>
   <End Role="Category" EntitySet="Categories">
     <Documentation>
         <Summary>This end of the relationship-instance describes the Category role for
AssociationSet CategoryProducts</Summary>
     </Documentation>
   </End>
   <End Role="Product" EntitySet="Products">
     <Documentation>
         <LongDescription>This end of the relationship-instance describes the Product role
for AssociationSet CategoryProducts</LongDescription>
     </Documentation>
   </End>
</AssociationSet>
```

Example 4: Documentation on <EntityType>, <Property> and <NavigationProperty>

```
<EntityType Name="Product">
   <Documentation>
      <Summary>Summary: EntityType named Product describes the content model for
Product</Summary>
      <LongDescription>LongDescription: The EntityType named Product describes the content
model for Product</LongDescription>
```

```
    </Documentation>
    <Key>
      <PropertyRef Name="ProductID" />
    </Key>
    <Property Name="ProductID" Type="Int32" Nullable="false">
        <Documentation>
            <Summary>Summary: This is the key property of EntityType Product</Summary>
            <LongDescription>LongDescription: This is the key property of EntityType
Product</LongDescription>
        </Documentation>
    </Property>
    <Property Name="ProductName" Type="String">
        <Documentation>
            <Summary>Summary: This property describes the name of the Product</Summary>
        </Documentation>
    </Property>
    <Property Name="QuantityPerUnit" Type="String">
        <Documentation>
          <LongDescription>LongDescription: This property describes the quantity per unit
corresponding to a product</LongDescription>
        </Documentation>
    </Property>
    <Property Name="PriceInfo" Nullable="false" Type="Self.PriceInfo" />
    <Property Name="StockInfo" Nullable="false" Type="Self.StockInfo" />
    <NavigationProperty Name="Category" Relationship="Self.CategoryProduct" FromRole="Product"
ToRole="Category">
        <Documentation>
            <Summary>This navigation property allows for traversing to Product-instances
associated with a Category-instance</Summary>
            <LongDescription> </LongDescription>
        </Documentation>
    </NavigationProperty>
</EntityType>
```

Example 5: Documentation on <Association> element

```
<Association Name="CategoryProduct">
   <Documentation>
       <Summary>Association CategoryProduct describes the participating end of the
CategoryProduct relationship</Summary>
   </Documentation>
   <End Role="Category" Type="Self.Category" Multiplicity="1">
      <Documentation>
          <Summary>This end of the relationship-instance describes the Category role for
Association CategoryProduct</Summary>
      </Documentation>
   </End>
   <End Role="Product" Type="Self.Product" Multiplicity="*">
      <Documentation>
          <LongDescription>This end of the relationship-instance describes the Product role
for Association CategoryProduct</LongDescription>
      </Documentation>
   </End>
</Association>
```

The following rules apply to the <Documentation> element.

- <Documentation> can contain any number of AnnotationAttributes. The full names of the AnnotationAttributes MUST NOT collide.

- <Documentation> can specify summary of the document inside a <Summary> element.

- <Documentation> can specify description of the documentation inside <LongDescription> element.

- Sub-elements for <Documentation> MUST appear in this sequence: <Summary>, <LongDescription>, AnnotationElement.

Element	Documentation		
Attributes	Name	Required	
	AnnotationAttribute	No	
Sub elements	Name	Occurrence	
		Min	Max
	Summary	0	1
	LongDescription	0	1
	AnnotationElement	0	Unbounded

All sub-elements MUST appear in the order indicated.

2.1.21 AnnotationElement

An AnnotationElement is a custom XML element applied to a CSDL element. The element and its sub-elements can belong to any XML namespace that is not in the list of reserved XML namespaces for CSDL. Consult the reference for each CSDL element within this document to determine whether an AnnotationElement can be used for that element.

Example:

```
<EntityType Name="Content">
  <Key>
    <PropertyRef Name="ID" />
  </Key>
  <Property Name="ID" Type="Guid" Nullable="false" />
  <Property Name="HTML" Type="String" Nullable="false" MaxLength="Max" Unicode="true"
     FixedLength="false" />
  <CLR:Attributes>
    <CLR:Attribute TypeName="System.Runtime.Serialization.DataContract"/>
    <CLR:Attribute TypeName="MyNamespace.MyAttribute"/>
  </CLR:Attributes>
  <RS:Security>
    <RS:ACE Principal="S-0-123-1321" Rights="+R+W"/>
    <RS:ACE Principal="S-0-123-2321" Rights="-R-W"/>
  </RS:Security>
</EntityType>
```

The following rules apply to Annotation elements:

- The namespace used in annotations MUST be declared or the namespace declaration MUST be in-lined with the annotation.

- Annotations MUST follow all other sub-elements. For example, when annotating an <EntityType> element the annotation element should follow all <Key>, <Property>, and <NavigationProperty> elements.

- More than one named Annotation can be defined per CSDL element.

- For a given CSDL element, Annotation element names can collide, so long as the combination of namespace + element name is unique for a particular element.

- Annotation is an XML element. It MUST contain a valid XML structure.

2.1.22 Model Functions

The <Function> element is used to define or declare a user function. These functions are defined as subelements of the <Schema> element.

Example:

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

- <Function> MUST have a **Name** attribute defined. The **Name** attribute is of type SimpleIdentifier. The **Name** attribute represents the name of this <Function>.

- <Function> MUST define a return type as an attribute or as a subelement.

- A <Function> MUST NOT contain both an attribute and a subelement defining the return type.

- A single <DefiningExpression> element can be defined for a given <Function>. A <DefiningExpression> is any expression that is intended to be the body of the function. The CSDL file format does not specify rules and restrictions regarding what language is to be used for specifying function bodies.

- All *Functionparameters* MUST be inbound.

- <Function> can contain any number of AnnotationAttributes. The full names of these AnnotationAttributes MUST NOT collide.

- Functions are declared as Global Items inside the <Schema> element.

- The function parameters and return type MUST be of the following types:

    - An EDMSimpleType or collection of EDMSimpleTypes.

    - An entity type or collection of entity types.

    - A complex type or collection of complex types.

    - A row type or collection of row types.

- A ref type or collection of ref types.

▪ <Function> can contain any number of <Parameter> elements.

▪ <Function> can contain any number of AnnotationElements.

▪ <Parameter>, <DefiningExpression>, <ReturnType>, and AnnotationElements can appear in any order.

Element	Function		
Attributes	Name	Required	
	Name	Yes	
	ReturnType	No	
	AnnotationAttribute	No	
Subelements	Name	Occurrence	
		Min	Max
	Documentation	0	1
	Choice	Parameter	0	Unbounded
		DefiningExpression	0	1
		ReturnType	0	1
		AnnotationElement	0	Unbounded

All subelements MUST appear in the order indicated. For all subelements within a given Choice, the subelements can be ordered arbitrarily.

2.1.23   Model Function Parameter

<Function> elements in CSDL only support inbound parameters. CSDL does not allow setting the <Function> <Parameter> mode; it is always set to Mode="In".

The type of a <Parameter> can be declared either as an attribute (example 1) or as a subelement (example 2):

Example 1

```
<Parameter Name="Age" Type="Edm.Int32"/>
```

Example 2

```
<Parameter Name="Owner">
    <TypeRef Name="Model.Person" />
</Parameter>
```

- <Parameter> MUST have a Name attribute defined. A **Name** attribute is of type SimpleIdentifier, and represents the name of this <Parameter>.

- <Parameter> MUST define the Type, either as an attribute or as a subelement.

- <Parameter> can define Facets if the type is an EDMSimpleType.

- <Parameter> can contain any number of AnnotationAttributes. The full name of these AnnotationAttributes MUST NOT collide.

- A function parameter MUST be one of the following types:

    - An EDMSimpleType or collection of EDMSimpleTypes.

    - An entity type or collection of entity types.

    - A complex type or collection of complex types.

    - A row type or collection of row types.

    - A ref type or collection of ref types.

- <Parameter> can contain a maximum of one <CollectionType> element.

- <Parameter> can contain a maximum of one <ReferenceType> element.

- <Parameter> can contain a maximum of one <RowType> element.

- <Parameter> can contain any number of AnnotationElements.

- AnnotationElements MUST be last in the sequence of subelements of a <Parameter>.

Element	Parameter		
Attributes	Name	Required	
	Name	Yes	
	Type	No	
	Facets	No	
	AnnotationAttribute	No	
Subelements	Name	Occurrence	
		Min	Max
	Choice — CollectionType	0	1
	Choice — ReferenceType	0	1
	Choice — RowType	0	1
	AnnotationElement	0	Unbounded

All subelements MUST appear in the order indicated. For all subelements within a given Choice, the subelements can be ordered arbitrarily.

2.1.24   CollectionType

If the type of the <Function> <Parameter> or <ReturnType> is a collection, it can be expressed either as an attribute (example 1) or by using subelement syntax (example 2).

Example 1

```
Type="Collection(Model.Person)"
```

Example 2

```
<Parameter Name="Owners">
```

```
                 <CollectionType>
                 <TypeRef Name="Model.Person" />
                 </CollectionType>
             </Parameter>
```

▪ <CollectionType> MUST define the type, either as an attribute or as a subelement.

▪ Attribute syntax MUST only be used if the collection is of a named type (that is, not of <RowType>).

▪ <CollectionType> can define Facets if the type is an EDMSimpleType.

▪ <CollectionType> can contain any number of AnnotationAttributes. The full name of these AnnotationAttributes MUST NOT collide.

▪ <CollectionType> can define one of the following as a sub-element:

  ▪ <CollectionType>

  ▪ <ReferenceType>

  ▪ <RowType>

  ▪ <TypeRef>

▪ <CollectionType> elements can contain any number of AnnotationElements.

▪ AnnotationElement MUST be last in the sequence of sub-elements of <CollectionType>.

Element	CollectionType			
Attributes	**Name**		**Required**	
	Type		No	
	Facets		No	
	AnnotationAttribute		No	
Subelements	**Name**		**Occurrence**	
			Min	Max
	Choice	CollectionType	0	1
		ReferenceType	0	1
		RowType	0	1
		TypeRef	0	1
	AnnotationElement		0	Unbounded

All sub-elements MUST appear in the order indicated. For all sub-elements within a given Choice, the sub-elements can be ordered arbitrarily.

2.1.25 TypeRef

<TypeRef> is used to reference an existing named type.

Example 1

```
<TypeRef Name="Model.Person" />
```

Example 2

```
<TypeRef Name="Edm.String" Nullable="True" MaxLength="50"/>
```

- <TypeRef> MUST have a **Type** attribute defined. The **Type** attribute defines the fully qualified name of the referenced type.

- <TypeRef> MUST be used to reference an existing named type. Named types include:

  - EntityType

  - ComplexType

  - Primitive Type

- <TypeRef> can define Facets if the type is an EDMSimpleType.

- <TypeRef> can contain any number of AnnotationAttributes. The full names of these AnnotationAttributes MUST NOT collide.

- <TypeRef> elements can contain at most one <Documentation> element.

- <TypeRef> elements can contain any number of AnnotationElements.

- AnnotationElement MUST be last in the sequence of sub-elements of <TypeRef>.

Element	TypeRef		
Attributes	Name	Required	
	Type	Yes	
	Facets	No	
	AnnotationAttribute	No	
Subelements	Name	Occurrence	
		Min	Max
	Documentation	0	1
	AnnotationElement	0	Unbounded

All sub-elements MUST appear in the order indicated.

2.1.26 ReferenceType

<ReferenceType> is used to specify the reference to an actual entity, either in the return type or in a parameter definition.

Example 1:

```
<ReferenceType Type="Model.Person" />
```

Example 2:

```
<ReturnType>
      <CollectionType>
            <ReferenceType Type="Model.Person" />
      </CollectionType>
</ReturnType>
```

- The <Type> attribute on a <ReferenceType> element MUST always be defined.

- The <Type> of the reference MUST always be of EntityType.

- <ReferenceType> can contain any number of AnnotationAttributes. The full names of these AnnotationAttributes MUST NOT collide.

- <ReferenceType> elements can contain at most one <Documentation> element.

- <ReferenceType> elements can contain any number of AnnotationElements.

- AnnotationElement MUST be last in the sequence of sub-elements of <ReferenceType>.

Element	ReferenceType		
Attributes	Name	Required	
	Type	Yes	
	AnnotationAttribute	No	
Subelements	Name	Occurrence	
		Min	Max
	Documentation	0	1
	AnnotationElement	0	Unbounded

All sub-elements MUST appear in the order indicated.

2.1.27 RowType

A <RowType> is an unnamed structure. <RowType> is always declared inline.

Example 1

```
<Parameter Name="Coordinate" Mode="In">
    <RowType>
        <Property Name="X" Type="int" Nullable="false"/>
        <Property Name="Y" Type="int" Nullable="false"/>
        <Property Name="Z" Type="int" Nullable="false"/>
    </RowType>
</Parameter>
```

Example 2

```
<ReturnType>
    <CollectionType>
        <RowType>
            <Property Name="X" Type="int" Nullable="false"/>
            <Property Name="Y" Type="int" Nullable="false"/>
            <Property Name="Z" Type="int" Nullable="false"/>
        </RowType>
    </CollectionType>
</ReturnType>
```

- <RowType> can contain any number of AnnotationAttributes. The full names of these AnnotationAttributes MUST NOT collide.

- <RowType> MUST contain at least one <Property> element.

- <RowType> can contain more than one <Property> element.

- The Type of a Property belonging to a RowType MUST be one of the following:

  - EntityType

  - ReferenceType

  - RowType

  - CollectionType

- <RowType> can contain any number of AnnotationElements.

Element	RowType			
Attributes	Name		Required	
	AnnotationAttribute		No	
Subelements	Name		Occurrence	
			Min	Max
	Choice	Property	1	Unbounded
		AnnotationElement	0	Unbounded

All subelements MUST appear in the order indicated. For all subelements within a given Choice, the subelements can be ordered arbitrarily.

2.1.28 RowType Property

One or more <Property> elements are used to describe the structure of <RowType>.

Example 1

```
<ReturnType>
    <CollectionType>
        <RowType>
                <Property Name="C" Type="Customer"/>
                <Property Name="Orders" Type="Collection(Order)"/>
        </RowType>
    </CollectionType>
</ReturnType>
```

Example 2

```
<ReturnType>
    <CollectionType>
        <RowType>
            <Property Name="Customer" Type="Customer"/>
            <Property Name="Orders">
                <CollectionType>
                    <RowType>
                        <Property Name="OrderNo" Type="Int32"/>
                        <Property Name="OrderDate" Type="Date"/>
                    <RowType>
                <CollectionType>
            </Property>
```

```
            </RowType>
        </CollectionType>
    </ReturnType>
```

- The Type of a Property belonging to a RowType MUST be one of the following:

  - EDMSimpleType

  - EntityType

  - ReferenceType

  - RowType

  - CollectionType

- \<RowType\> MUST have a **Name** attribute defined. The **Name** attribute is of type SimpleIdentifier. The **Name** attribute represents the name of this \<Property\>.

- \<RowType\> MUST define a type as an attribute or as a subelement.

- \<RowType\> MUST NOT contain both an attribute and a subelement defining the type.

- \<RowType\> can define Facets if the type is an EDMSimpleType.

- \<RowType\> can contain any number of AnnotationAttributes. The full names of these AnnotationAttributes MUST NOT collide.

- \<RowType\> can contain any number of AnnotationElements.

- AnnotationElements MUST be last in the sequence of subelements of \<RowType\>.

Element	Property	
Attributes	Name	Required
	Name	Yes
	Type	No
	Facets	No
	AnnotationAttribute	No

	Name	Occurrence	
		Min	Max
Subelements	Choice — CollectionType	0	1
	Choice — ReferenceType	0	1
	Choice — RowType	0	1
	AnnotationElement	0	Unbounded

All subelements MUST appear in the order indicated. For all subelements within a given Choice, the subelements can be ordered arbitrarily.

2.1.29 Function ReturnType

<ReturnType> describes the shape of data returned from a <Function>. The return type of a function can be declared as a **ReturnType** attribute on a <Function> (example 1), or as a subelement (example 2).

Example 1

```
<Function Name="GetAge" ReturnType="Edm.Int32">
```

Example 2

```
<Function Name="GetAge">
    <ReturnType Type="Edm.Int32" />
</Function>
```

- <ReturnType> MUST define type declaration as an attribute or as a subelement.

- <ReturnType> MUST NOT contain both an attribute and a subelement defining the type.

- <ReturnType> can contain any number of AnnotationAttributes. The full names of these AnnotationAttributes MUST NOT collide.

- The return type of <Function> MUST be one of the following:

    - An EDMSimpleType or collection of EDMSimpleTypes.

    - An entity type or collection of entity types.

    - A complex type or collection of complex types.

    - A row type or collection of row types.

    - <ReturnType> can contain a maximum of one <CollectionType> element.

    - <ReturnType> can contain a maximum of one <ReferenceType> element.

    - <ReturnType> can contain a maximum of one <RowType> element.

    - A ref type or collection of ref types.

- <ReturnType> can contain any number of AnnotationElements.

- AnnotationElements MUST be last in the sequence of subelements of <ReturnType>.

Element	ReturnType			
Attributes	Name		Required	
	Type		No	
	AnnotationAttribute		No	
Subelements	Name		Occurrence	
			Min	Max
	Choice	CollectionType	0	1
		ReferenceType	0	1
		RowType	0	1
	AnnotationElement		0	Unbounded

All sub-elements MUST appear in the order indicated. For all subelements within a given Choice, the subelements can be ordered arbitrarily.

2.2 Attributes

2.2.1 EDMSimpleType

The Entity Data Model (EDM) defines an abstract type system that defines the primitive types listed in the following sections.

2.2.1.1 Commonly Applicable Facets

2.2.1.1.1 Nullable

The Nullable facet is a Boolean, which indicates whether the **Type** can be null.

2.2.1.1.2 Default

The Default facet is a string. Valid values for this facet depend upon the type being referenced.

2.2.1.2 Binary

The binary data type is used to represent fixed or variable length binary data.

2.2.1.2.1 Facets

The EDM simple type facets applicable for this type are FixedLength and MaxLength.

2.2.1.2.1.1 MaxLength

The MaxLength facet is a positive integer with value ranging from 1 to 2^31. The maximum size of the declared binary data type value is specified by the value of the MaxLength facet.

2.2.1.2.1.2 FixedLength

The FixedLength facet is a Boolean that indicates whether the length can vary.

2.2.1.3 Boolean

The Boolean data type is used to represent the mathematical concept of binary valued logic. There are no applicable facets for this type.

2.2.1.4 DateTime

The DateTime type represents date and time with values ranging from 12:00:00 midnight, January 1, 1753 A.D. through 11:59:59 P.M, December 31, 9999 A.D..

2.2.1.4.1 Facets

2.2.1.4.1.1 Precision

The Precision facet indicates the degree of granularity of the DateTime in fractions of a second, based on the number of decimal places supported. The actual values allowed will depend on the data provider. As an example, the Microsoft database allows a Precision of 3, meaning that the granularity supported is milliseconds.

2.2.1.5 Time

The Time type represents the time of day with values ranging from 0:00:00.x to 23:59:59.y, where x and y depend upon the precision.

2.2.1.5.1 Facets

2.2.1.5.1.1 Precision

The Precision facet indicates the degree of granularity of the Time in fractions of a second, based on the number of decimal places supported. The actual values allowed will depend on the data provider. As an example, the Microsoft database allows a Precision of 3, meaning that the granularity supported is milliseconds. If the Precision is 3, the minimum value for time is 0:00:00:001 and the maximum is 23:59:59.999.

2.2.1.6 DateTimeOffset

The DateTimeOffset type represents date and time as an Offset in minutes from GMT, with values ranging from 12:00:00 midnight, January 1, 1753 A.D. through 11:59:59 P.M, December 9999 A.D.

2.2.1.6.1 Facets

2.2.1.6.1.1 Precision

The Precision facet indicates the degree of granularity of the DateTimeOffset in fractions of a second, based on the number of decimal places supported. For example, a Precision of 3 means the granularity supported is milliseconds.

2.2.1.7 Decimal

The Decimal type represents numeric values with fixed precision and scale. The required precision and scale can be specified using its optional Precision and Scale facets. This type can describe a numeric value ranging from negative $10^{255} + 1$ to positive $10^{255} -1$.

2.2.1.7.1 Facets

2.2.1.7.1.1 Precision

This is a positive integer that specifies the maximum number of decimal digits that an instance of the decimal type can have, both to the left and to the right of the decimal point. Possible values for Precision are 1, 2, or 3.

2.2.1.7.1.2 Scale

This is a positive integer that specifies the maximum number of decimal digits to the right of the decimal point that an instance of this type can have. The Scale value can range from 0 through the specified Precision value. The default Scale is 0.

2.2.1.8 Single

The Single type represents a floating point number with 7 digits precision that can represent values with approximate range of ± 1.18e -38 through ± 3.40e +38.

2.2.1.9 Double

The Double type represents a floating point number with 15 digits precision that can represent values with approximate range of ± 2.23e -308 through ± 1.79e +308.

2.2.1.10 Guid

This **Guid** type, as specified in [RFC4122], represents a 16-byte (128-bit) unique identifier value.

2.2.1.11 SByte

The SByte type represents a signed 8-bit integer value.

2.2.1.12 Int16

The Int16 type represents a signed 16-bit integer value.

2.2.1.13 Int32

The Int32 type represents a signed 32-bit integer value.

2.2.1.14 Int64

The Int64 type represents a signed 64-bit integer value.

2.2.1.15 Byte

The Byte type represents an unsigned 8-bit integer value.

2.2.1.16 String

The String type represents fixed or variable length character data. The EDMSimpleType facets applicable to String type are described below.

2.2.1.16.1 Facets

The EDM simple type facets applicable for this type are Unicode, Collation, FixedLength and MaxLength. The other facets, Unicode and Collation, are optional.

2.2.1.16.1.1 Unicode

The Unicode facet is a Boolean value. This value, when set to true, dictates the string type that an instance will store. By default, **UNICODE** characters are used, otherwise standard ASCII encoding is used. The default value for this facet is true.

Note The string data type does not support the kind of **UNICODE** to be specified, leaving it to the concrete type systems hosting EDM to choose the appropriate **UNICODE** type.

2.2.1.16.1.2 FixedLength

The FixedLength facet is a Boolean value. The value indicates whether the store requires a string to be fixed length or not (that is, in SqlServer setting this facet to true would require a fixed-length field (char or nchar) instead of variable-length (varchar or nvarchar)).

2.2.1.16.1.3 MaxLength

The MaxLength facet specifies the maximum length of an instance of the string type. For Unicode equal to True, the Maxlength can range from 1 to 2^{30}, or if False, 1 to 2^{31}.

2.2.1.16.1.4 Collation

The Collation facet is a string value that specifies the collating sequence (or sorting sequence) to be used for performing comparison and ordering operations over string values.

	Binary
	Boolean
	Byte
	DateTime
	DateTimeOffset
EDMSimpleType (restriction base="xs:string")	Time

EDMSimpleType (restriction base="xs:string")	Binary
	Decimal
	Double
	Single
	Guid
	Int16
	Int32
	Int64
	String
	SByte

2.2.2 Action

Action can either be "Cascade" or "None".

The Cascade action implies that the operation to delete an Entity should delete the relationship instance and then apply the action on the entity-instance at the other end of the relationship. For instance, when a Customer is deleted, delete all Orders belonging to that Customer.

	Cascade
Action	None

2.2.3 Multiplicity

The **Multiplicity** of a relationship describes the cardinality or number of instances of an <EntityType> that can be associated with the instances of another <EntityType>. The possible types of multiplicity are: one-to-one, one-to-many, zero-one to one, zero-one to many, and many-to-many.

	0..1
	1
Multiplicity	*

2.2.4 ConcurrencyMode

ConcurrencyMode is a special facet which can be applied to any primitive Entity Data Model (EDM) type. Possible values are "None", which is the default, and "Fixed".

When used on an <EntityType> property, it indicates that the value of that declared property should be used for optimistic concurrency checks. Essentially, declared properties marked with a "Fixed" ConcurrencyMode become part of a ConcurrencyToken.

The following rules apply to **ConcurrencyMode**:

- The property's type MUST be a simple type; it can't be applied to properties of a ComplexType.

- The property MUST be a declared property.

	None
ConcurrencyMode	Fixed

2.2.5 QualifiedName

QualifiedName is a string-based representation of the name of the element or attribute.

The pattern below represents the allowed identifiers for **QualifiedName**:

Pattern:

```
Value="[\p{L}\p{Nl}][\p{L}\p{Nl}\p{Nd}\p{Mn}\p{Mc}\p{Pc}\p{Cf}]{0,}(\.[\p{L}\p{Nl}][\p{L}\p{Nl}\p{Nd}\p{Mn}\p{Mc}\p{Pc}\p{Cf}]{0,}){0,}"
```

2.2.6 SimpleIdentifier

SimpleIdentifier is a string-based representation. The maximum length of the identifier MUST be less than 480.

The below pattern represents the allowed identifiers in ECMA specification:

Pattern:

```
value="[\p{L}\p{Nl}][\p{L}\p{Nl}\p{Nd}\p{Mn}\p{Mc}\p{Pc}\p{Cf}]{0,}"
```

2.2.7 AnnotationAttribute

An AnnotationAttribute is a custom XML attribute applied to a CSDL Element. The attribute can belong to any XML namespace (as defined in [XMLNS-2ED]) that is not in the list of reserved XML namespaces for CSDL. Consult the reference for each CSDL element within this document to determine whether AnnotationAttribute may be used for that element.

2.2.8 OpenType

OpenType is a facet which can be applied to any <EntityType>. Possible values are "false", which is the default, and "true".

<EntityType> elements marked with OpenType="false" or <EntityType> elements which do not explicitly include an OpenType attribute indicate that the element defines an **EntityType**. <EntityType> elements marked with OpenType="true" indicate that the element defines an **OpenEntityType**.

	true
OpenType	false

3 Structure Examples

The following example shows a CSDL that defines:

- Customer, Order, and Product entity types.

- Association (CustomerOrder) that associates Customer and Order entity types.

- SalesOrder entity type that has Order as the BaseType.

- Address complex type.

```
<Schema xmlns="http://schemas.microsoft.com/ado/2008/09/edm" Namespace="Model1" Alias="Self">
  <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>
  <EntityType Name="Customer">
    <Key>
      <PropertyRef Name="CustomerId" />
    </Key>
    <Property Name="CustomerId" Type="Int32" Nullable="false" />
    <Property Name="FirstName" Type="String" Nullable="true" />
    <Property Name="LastName" Type="String" Nullable="true" />
    <Property Name="AccountNumber" Type="Int32" Nullable="true" />
    <Property Name="Address" Type="Self.Address" Nullable="false" />
    <NavigationProperty Name="Orders" Relationship="Model1.CustomerOrder" FromRole="Customer"
ToRole="Order" />
  </EntityType>
  <EntityType Name="Order">
    <Key>
      <PropertyRef Name="OrderId" />
    </Key>
    <Property Name="OrderId" Type="Int32" Nullable="false" />
    <Property Name="OrderDate" Type="Int32" Nullable="true" />
    <Property Name="Description" Type=" String" Nullable="true" />
    <NavigationProperty Name="Customer" Relationship="Model1.CustomerOrder" FromRole="Order"
ToRole="Customer" />
  </EntityType>
  <EntityType Name="SalesOrder" BaseType="Self.Order">
    <Property Name="Paid" Type="Boolean" Nullable="false" />
  </EntityType>
  <EntityType OpenType="true" Name="Product">
    <Key>
      <PropertyRef Name="ProductId" />
    </Key>
    <Property Name="ProductId" Type="Int32" Nullable="false" />
    <Property Name="Name" Type="String" Nullable="false" />
    <Property Name="Description" Type="String" Nullable="true" />
  </EntityType>
  <Association Name="CustomerOrder">
    <End Type="Model1.Customer" Role="Customer" Multiplicity="1" />
```

```
      <End Type="Model1.Order" Role="Order" Multiplicity="*" />
    </Association>
    <ComplexType Name="Address">
      <Property Name="Street" Type="String" Nullable="false" />
      <Property Name="City" Type="String" Nullable="false" />
      <Property Name="State" Type="String" Nullable="false" />
      <Property Name="Zip" Type="String" Nullable="false" />
    </ComplexType>
  </Schema>
```

4   Security Considerations

None.

5   Appendix A: Product Behavior

This document specifies version-specific details in the Microsoft® .NET Framework. For information about which versions of .NET Framework are available in each released Microsoft Windows® product or as supplemental software, see **.NET Framework**.

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

▪   Microsoft® .NET Framework 3.5

▪   Microsoft® .NET Framework 4.0

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

6 Appendix B: Differences Between CSDL Version 1.0 and CSDL Version 1.1

CSDL version 1.1 is a superset of CSDL version 1.0.

This section outlines the differences between CSDL version 1.0 and CSDL version 1.1.

For CSDL version 1.0, the following rules apply.

- **ComplexType** must not define an **Abstract** attribute.

- **ComplexType** must not define a **BaseType** attribute.

- **ReturnType** for a <FunctionImport> must be a Collection.

- **ReturnType** for a <FunctionImport> must not be a Collection of **ComplexType**.

- <Property> must not define a **CollectionKind** attribute.

- <Property> of type **ComplexType** must not be Nullable.

7   Appendix C: Differences Between CSDL Version 1.1 and CSDL Version 1.2

CSDL version 1.2 is a superset of CSDL version 1.1.

This section outlines the differences between CSDL version 1.1 and CSDL version 1.2.

For CSDL version 1.1, the following rules apply:

- <EntityType> must not define an <OpenType> attribute.

8   Appendix D: Differences Between CSDL Version 1.2 and CSDL Version 2.0

CSDL version 2.0 is a superset of CSDL version 1.2.

This section outlines the differences between CSDL version 1.2 and CSDL version 2.0.

For CSDL version 1.2, the following rules apply:

- <Schema> must not contain any <Function> sub-elements

- Entity <Key> must not define any AnnotationElement elements

- In CSDL 1.2 and lower versions binary data type is not supported for defining <Key>.

- Entity <PropertyRef> must not define any AnnotationElement elements

- <ReferentialConstraints>, <Role> must not define any AnnotationElement elements

- <EntityContainer> must not define any AnnotationElement elements

- <FunctionImport> must not define any AnnotationElement elements

- <ReferentialConstraint> must only exist between the key properties of associated entities

Release: Friday, February 4, 2011

9   Change Tracking

This section identifies changes that were made to the [MC-CSDL] protocol document between the January 2011 and February 2011 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.

- An extensive rewrite, addition, or deletion of major portions of content.

- The removal of a document from the documentation set.

- Changes made for template compliance.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.

- Content updated.

- Content removed.

- New product behavior note added.

- Product behavior note updated.

- Product behavior note removed.

- New protocol syntax added.

- Protocol syntax updated.

- Protocol syntax removed.

- New content added due to protocol revision.

- Content updated due to protocol revision.

- Content removed due to protocol revision.

- New protocol syntax added due to protocol revision.

- Protocol syntax updated due to protocol revision.

- Protocol syntax removed due to protocol revision.

- New content added for template compliance.

- Content updated for template compliance.

- Content removed for template compliance.

- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated.**

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.

- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
2.1.28 RowType Property	55602 Added EDMSimpleType to the Type list for the RowType property.	N	Content updated.
2.2.1.2.1 Facets	58288 Changed fixedLength to FixedLength and maxLength to MaxLength.	N	Content updated.
2.2.1.2.1.1 MaxLength	58288 Changed maxLength to MaxLength.	N	Content updated.
2.2.1.2.1.2 FixedLength	58288 Changed fixedLength to FixedLength.	N	Content updated.
2.2.1.16.1 Facets	58288 Changed unicode to Unicode, fixedLength to FixedLength, maxLength to MaxLength, and collation to Collation.	N	Content updated.
2.2.1.16.1.1 Unicode	58288 Changed "unicode" to "Unicode".	N	Content updated.
2.2.1.16.1.2 FixedLength	58288 Changed fixedLength to FixedLength.	N	Content updated.
2.2.1.16.1.3 MaxLength	58288 Changed unicode to Unicode and maxLength to MaxLength.	N	Content updated.
2.2.1.16.1.4 Collation	58288 Changed collation to Collation.	N	Content updated.

Release: Friday, February 4, 2011

10 Index