OData Common Schema Definition Language (CSDL) XML Representation Version 4.02

Committee Specification Draft 01

14 July 2023

This stage:

https://docs.oasis-open.org/odata/odata-csdl-json/v4.02/csd01/odata-csdl-xml-v4.02-csd01.md (Authoritative)
https://docs.oasis-open.org/odata/odata-csdl-json/v4.02/csd01/odata-csdl-xml-v4.02-csd01.html
https://docs.oasis-open.org/odata/odata-csdl-json/v4.02/csd01/odata-csdl-xml-v4.02-csd01.pdf

Previous stage:

N/A

Latest stage:

https://docs.oasis-open.org/odata/odata-csdl-json/v4.02/odata-csdl-json-v4.02.md (Authoritative)
https://docs.oasis-open.org/odata/odata-csdl-json/v4.02/odata-csdl-json-v4.02.html
https://docs.oasis-open.org/odata/odata-csdl-json/v4.02/odata-csdl-json-v4.02.pdf

Technical Committee:

OASIS Open Data Protocol (OData) TC

Chairs:

Ralf Handl (ralf.handl@sap.com), SAP SE
Michael Pizzo (mikep@microsoft.com), Microsoft

Editors:

Ralf Handl (ralf.handl@sap.com), SAP SE
Michael Pizzo (mikep@microsoft.com), Microsoft
Heiko Theißen (heiko.theissen@sap.com), SAP SE

Additional artifacts:

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

- XML schemas: *OData EDMX XML Schema and OData EDM XML Schema*. Latest stage: https://docs.oasis-open.org/odata/odata-csdl-xml/v4.01/os/schemas/

Related work:

This specification replaces or supersedes:

- *OData Common Schema Definition Language (CSDL) JSON Representation Version 4.01*. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. OASIS Standard. Latest stage: https://docs.oasis-open.org/odata/odata-csdl-json/v4.01/odata-csdl-json-v4.01.html.

This specification is related to:

- *OData Version 4.02*. Edited by Michael Pizzo, Ralf Handl, and Heiko Theißen. A multi-part Work Product that includes:
  - *OData Version 4.02 Part 1: Protocol*. Latest stage: https://docs.oasis-open.org/odata/odata/v4.02/odata-v4.02-part1-protocol.html
  - *OData Version 4.02 Part 2: URL Conventions*. Latest stage: https://docs.oasis-open.org/odata/odata/v4.02/odata-v4.02-part2-url-conventions.html
  - *ABNF components: OData ABNF Construction Rules Version 4.01 and OData ABNF Test Cases*. https://docs.oasis-open.org/odata/odata/v4.01/os/abnf/
- *OData Vocabularies Version 4.0*. Edited by Michael Pizzo, Ralf Handl, and Ram Jeyaraman. Latest stage: https://docs.oasis-open.org/odata/odata-vocabularies/v4.0/odata-vocabularies-v4.0.html
- *OData Common Schema Definition Language (CSDL) JSON Representation Version 4.01*. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. Latest stage: https://docs.oasis-open.org/odata/odata-csdl-json/v4.01/odata-csdl-json-v4.01.html
- *OData JSON Format Version 4.01*. Edited by Ralf Handl, Mike Pizzo, and Mark Biamonte. Latest stage: https://docs.oasis-open.org/odata/odata-json-format/v4.01/odata-json-format-v4.01.html

Abstract:

OData services are described by an Entity Model (EDM). The Common Schema Definition Language (CSDL) defines specific representations of the entity data model exposed by an OData service, using XML, JSON, and other formats. This document (OData CSDL JSON Representation) specifically defines the JSON representation of CSDL.

Status:

This document was last revised or approved by the OASIS Open Data Protocol (OData) TC on the above date. The level of approval is also listed above. Check the "Latest stage" location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=odata#technical.

TC members should send comments on this specification to the TC's email list. Others should send comments to the TC's public comment list, after subscribing to it by following the instructions at the "Send A Comment" button on the TC's web page at https://www.oasis-open.org/committees/odata/.

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

Key words:

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 **[RFC2119]** and **[RFC8174]** when, and only when, they appear in all capitals, as shown here.

Citation format:

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

[OData-CSDL-JSON-v4.02]

OData Common Schema Definition Language (CSDL) JSON Representation Version 4.02. Edited by Ralf Handl, Michael Pizzo, and Heiko Theißen. 14 July 2023. OASIS Committee Specification Draft 01. https://docs.oasis-open.org/odata/odata-csdl-json/v4.02/csd01/odata-csdl-xml-v4.02-csd01.html. Latest stage: https://docs.oasis-open.org/odata/odata-csdl-json/v4.02/odata-csdl-json-v4.02.html.

Notices

Table of Contents

1 Introduction

OData services are described in terms of an Entity Model. The Common Schema Definition Language (CSDL) defines a representation of the entity model exposed by an OData service using the JavaScript Object Notation (JSON), see **[RFC8259]**.

This format is based on the OpenUI5 OData V4 Metadata JSON Format, see **[OpenUI5]**, with some extensions and modifications made necessary to fully cover OData CSDL Version 4.01.

1.1 Changes from earlier Versions

1.2 Glossary

1.2.1 Definitions of terms

1.2.2 Acronyms and abbreviations

1.2.3 Document conventions

Keywords defined by this specification use .

Some sections of this specification are illustrated with non-normative examples.

Example 1: text describing an example uses this paragraph style

```
Non-normative examples use this paragraph style.
```

All examples in this document are non-normative and informative only. Examples labeled with ⚠ contain advanced concepts or make use of keywords that are defined only later in the text, they can be skipped at first reading.

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

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

All other text is normative unless otherwise labeled.

Here is a customized command line which will generate HTML from this markdown file (named). Line breaks are added for readability only:

```
pandoc -f gfm+tex_math_dollars+fenced_divs
       -t html
       -o odata-csdl-xml-v4.02-csd01.html
       -c styles/markdown-styles-v1.7.3b.css
       -c styles/odata.css
       -s
       --mathjax
       --eol=lf
       --wrap=none
       --metadata pagetitle="OData Common Schema Definition Language (CSDL) XML
```



```
Representation Version 4.02"
       odata-csdl-xml-v4.02-csd01.md
```

This uses pandoc 3.1.2 from https://github.com/jgm/pandoc/releases/tag/3.1.2.

2 XML Representation

OData CSDL XML is a full representation of the OData Common Schema Definition Language in the Extensible Markup Language (XML) 1.1 (Second Edition) **[XML-1.1]** with further building blocks from the W3C XML Schema Definition Language (XSD) 1.1 as described in **[XML-Schema-1]** and **[XML-Schema-2]**.

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

2.1 Requesting the XML Representation

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

Alternatively, this representation can be requested using the header with the media type , optionally followed by media type parameters.

If specified, overrides any value specified in the header.

The response MUST contain the header with a value of , optionally followed by media type parameters.

This specification does not define additional parameters for the media type .

2.2 XML Namespaces

In addition to the default XML namespace, the elements and attributes used to describe the entity model of an OData service are defined in one of the following namespaces.

2.2.1 Namespace EDMX

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

-

Prior versions of OData used the following namespace for EDMX:

- EDMX version 1.0:

They are non-normative for this specification.

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

2.2.2 Namespace EDM

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

-

Prior versions of CSDL used the following namespaces for EDM:

- CSDL version 1.0:
- CSDL version 1.1:
- CSDL version 1.2:
- CSDL version 2.0:
- CSDL version 3.0:

They are non-normative for this specification.

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

2.3 XML Schema Definitions

This specification contains normative XML schemas for the EDMX and EDM namespaces; see **[OData-EDMX]** and **[OData-EDM]**

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

2.4 XML Document Order

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

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

3 Entity Model

An OData service exposes a single entity model. This model may be distributed over several schemas, and these schemas may be distributed over several documents.

A service is defined by a single CSDL document which can be accessed by sending a request to . This document is called the metadata document. It MAY reference other CSDL documents.

The metadata document contains a single entity container that defines the resources exposed by this service. This entity container MAY extend an entity container defined in a referenced document.

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

The *scope* of a CSDL document is the document itself and all schemas included from directly referenced documents. All entity types, complex types and other named model elements *in scope* (that is, defined in the document itself or a schema of a directly referenced document) can be accessed from a referencing document by their qualified names. This includes the built-in primitive and abstract types.

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

Note: referencing documents is not recursive. Only named model elements defined in directly referenced documents can be used within the schema. However, those elements may in turn include or reference model elements defined in schemas referenced by their defining schema.

3.1 Nominal Types

A nominal type has a name that MUST be a simple identifier. Nominal types are referenced using their qualified name. The qualified type name MUST be unique within a model as it facilitates references to the element from other parts of the model.

Names are case-sensitive, but service authors SHOULD NOT choose names that differ only in case.

3.2 Structured Types

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

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

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

3.3 Primitive Types

Structured types are composed of other structured types and primitive types. OData defines the following primitive types:

Type	Meaning
	Binary data
	Binary-valued logic
	Unsigned 8-bit integer

Type	Meaning
	Date without a time-zone offset
	Date and time with a time-zone offset, no leap seconds
	Numeric values with decimal representation
	IEEE 754 binary64 floating-point number (15-17 decimal digits)
	Signed duration in days, hours, minutes, and (sub)seconds
	16-byte (128-bit) unique identifier
	Signed 16-bit integer
	Signed 32-bit integer
	Signed 64-bit integer
	Signed 8-bit integer
	IEEE 754 binary32 floating-point number (6-9 decimal digits)
	Binary data stream
	Sequence of characters
	Clock time 00:00-23:59:59.999999999999
	Abstract base type for all Geography types
	A point in a round-earth coordinate system
	Line string in a round-earth coordinate system
	Polygon in a round-earth coordinate system
	Collection of points in a round-earth coordinate system
	Collection of line strings in a round-earth coordinate system
	Collection of polygons in a round-earth coordinate system
	Collection of arbitrary Geography values
	Abstract base type for all Geometry types
	Point in a flat-earth coordinate system
	Line string in a flat-earth coordinate system

Type	Meaning
	Polygon in a flat-earth coordinate system
	Collection of points in a flat-earth coordinate system
	Collection of line strings in a flat-earth coordinate system
	Collection of polygons in a flat-earth coordinate system
	Collection of arbitrary Geometry values

and follow **[XML-Schema-2]** and use the proleptic Gregorian calendar, allowing the year (equivalent to 1 BCE) and negative years (year being equivalent to 2 BCE etc.). The supported date range is service-specific and typically depends on the underlying persistency layer, e.g. SQL only supports years to .

with a ____ value of , , and allow the special numeric values , , and .

is a primitive type that can be used as a property of an entity type or complex type, the underlying type for a type definition, or the binding parameter or return type of an action or function. , or a type definition whose underlying type is , cannot be used in collections or for non-binding parameters to functions or actions.

Some of these types allow facets, defined in section "Type Facets".

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

3.4 Built-In Abstract Types

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

-
-
-
-

Conceptually, these are the abstract base types for primitive types (including type definitions and enumeration types), complex types, entity types, or any type or collection of types, respectively, and can be used anywhere a corresponding concrete type can be used, except:

-
  - cannot be used as the type of a singleton in an entity container because it doesn't define a structure, which defeats the purpose of a singleton.
  - cannot be used as the type of an entity set because all entities in an entity set must have the same key fields to uniquely identify them within the set.
  - cannot be the base type of an entity type or complex type.

-
  - cannot be the base type of an entity type or complex type.
-
  - cannot be used as the type of a key property of an entity type or as the underlying type of an enumeration type.
  - cannot be used as the underlying type of a type definition in a CSDL document with a version of .
  - can be used as the underlying type of a type definition in a CSDL document with a version of or greater.
-
  - cannot be returned in a payload with an header of . Services should treat untyped properties as dynamic properties in payloads.
  - cannot be used as the type of a key property of an entity type.
  - cannot be the base type of an entity type or complex type.
  - cannot be used as the underlying type of a type definition or enumeration type.
-
  - cannot be used as the type of a property or term.
  - cannot be used as the type of a parameter or the return type of an action or function.
-
  - cannot be returned in a payload with an header of . Services should treat untyped properties as dynamic properties in payloads.

3.5 Built-In Types for defining Vocabulary Terms

Vocabulary terms can, in addition, use

-
-
-
- (or)
- (any model element, including , , and)

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

3.6 Annotations

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

A model element MUST NOT specify more than one annotation for a given combination of term and qualifier.

4 CSDL XML Document

Element

The element is the root element of a CSDL XML document. It MUST contain the attribute and it MUST contain exactly one element.

It MAY contain elements to reference other CSDL documents.

Attribute

The attribute specifies the OData protocol version of the service. For OData 4.0 responses the value of this attribute MUST be For OData 4.01 responses the value of this attribute MUST be Services MUST return an OData 4.0 response if the request was made with an header with a value of .

Element

The element MUST contain one or more elements which define the schemas exposed by the OData service.

Example 2:

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

4.1 Reference

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

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

A reference MAY be annotated.

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

Element

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

The element MUST contain the attribute, and it MUST contain at least one or child element.

It MAY contain elements.

Attribute

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

Example 3: references to other CSDL documents

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

4.2 Included Schema

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

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

When including a schema, a simple identifier value MAY be specified as an alias for the schema that is used in qualified names instead of the namespace. For example, an alias of might be assigned to the namespace . An alias-qualified name is resolved to a fully qualified name by examining aliases for included schemas and schemas defined within the document.

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

Aliases are document-global, so all schemas defined within or included into a document MUST have different aliases, and aliases MUST differ from the namespaces of all schemas defined within or included into a document.

The alias MUST NOT be one of the reserved values , , , or .

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

> **Element**
>
> The element specifies a schema to include from the referenced CSDL document. It MUST provide the attribute and it MAY provide the attribute.
>
> It MAY contain elements.
>
> **Attribute**
>
> The value of is the namespace of a schema defined in the referenced CSDL document.
>
> **Attribute**
>
> The value of is a simple identifier that can be used in qualified names instead of the namespace.

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

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

4.3 Included Annotations

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

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

interest for the consumer.

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

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

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

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

> **Element**
>
> The element specifies the annotations to include from the referenced CSDL document. If no element is specified, a client MAY ignore all annotations in the referenced document that are not explicitly used in an expression of the referencing document.
>
> The element MUST provide the attribute, and it MAY provide the and attribute.
>
> **Attribute**
>
> The value of is a namespace.
>
> **Attribute**
>
> The value of is a simple identifier.
>
> **Attribute**
>
> The value of is a namespace.

Example 5: reference documents that contain annotations

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx"
           Version="4.0">
  <edmx:Reference Uri="http://odata.org/ann/b">
```



```
    <edmx:IncludeAnnotations TermNamespace="org.example.validation" />
    <edmx:IncludeAnnotations TermNamespace="org.example.display"
                             Qualifier="Tablet" />
    <edmx:IncludeAnnotations TermNamespace="org.example.hcm"
                             TargetNamespace="com.example.Sales" />
    <edmx:IncludeAnnotations TermNamespace="org.example.hcm"
                             Qualifier="Tablet"
                             TargetNamespace="com.example.Person" />
  </edmx:Reference>
  <edmx:DataServices>…</edmx:DataServices>
</edmx:Edmx>
```

The following annotations from *are included:*

- *Annotations that use a term from the* *namespace, and*
- *Annotations that use a term from the* *namespace and specify a* *qualifier and*
- *Annotations that apply a term from the* *namespace to an element of the* *namespace and*
- *Annotations that apply a term from the* *namespace to an element of the* *namespace and specify a* *qualifier.*

5 Schema

One or more schemas describe the entity model exposed by an OData service. The schema acts as a namespace for elements of the entity model such as entity types, complex types, enumerations and terms.

A schema is identified by a namespace. Schema namespaces MUST be unique within the scope of a document and SHOULD be globally unique. A schema cannot span more than one document.

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

Names are case-sensitive, but service authors SHOULD NOT choose names that differ only in case.

The namespace MUST NOT be one of the reserved values , , , or .

> **Element**
>
> The element defines a schema. It MUST contain the attribute and it MAY contain the attribute.
>
> It MAY contain elements , , , , , , , , , or .
>
> **Attribute**
>
> The value of is the namespace of the schema

5.1 Alias

A schema MAY specify an alias which MUST be a simple identifier.

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

Aliases are document-global, so all schemas defined within or included into a document MUST have different aliases, and aliases MUST differ from the namespaces of all schemas defined within or included into a document. Aliases defined by a schema can be used throughout the containing document and are not restricted to the schema that defines them.

The alias MUST NOT be one of the reserved values , , , or .

> **Attribute**
>
> The value of is a simple identifier.

Example 6: schema with an alias and a description for the schema

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

5.2 Annotations with External Targeting

Element

The element is used to apply a group of annotations to a single model element. It MUST contain the attribute and it MAY contain the attribute.

It MUST contain at least one element.

Attribute

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

Attribute

The value of is a simple identifier.

Example 7: annotations should only be applied to tablet devices

```
<Annotations Target="org.example.Person" Qualifier="Tablet">
  <Annotation Term="Core.Description" String="Dummy" />
  ...
</Annotations>
```

6 Entity Type

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

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

An entity type can define two types of properties. A structural property is a named reference to a primitive, complex, or enumeration type, or a collection of primitive, complex, or enumeration types. A navigation property is a named reference to another entity type or collection of entity types.

All properties MUST have a unique name within an entity type. Properties MUST NOT have the same name as the declaring entity type. They MAY have the same name as one of the direct or indirect base types or derived types.

> **Element**
>
> The element MUST contain the attribute, and it MAY contain the , , , and attributes.
>
> It MAY contain and elements describing the properties of the entity type.
>
> It MAY contain one element.
>
> It MAY contain elements.

> **Attribute**
>
> The value of is the entity type's name.

Example 8: a simple entity type

```
<EntityType Name="Employee">
  <Key>
    <PropertyRef Name="ID" />
  </Key>
  <Property Name="ID" Type="Edm.String" Nullable="false" />
  <Property Name="FirstName" Type="Edm.String" Nullable="false" />
  <Property Name="LastName" Type="Edm.String" Nullable="false" />
  <NavigationProperty Name="Manager" Type="self.Manager" />
</EntityType>
```

6.1 Derived Entity Type

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

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

An entity type MUST NOT introduce an inheritance cycle by specifying a base type.

> **Attribute**
>
> The value of is the qualified name of the base type.

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

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

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

6.2 Abstract Entity Type

An entity type MAY indicate that it is abstract and cannot have instances.

For OData 4.0 responses a non-abstract entity type MUST define a key or derive from a base type with a defined key.

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

> **Attribute**
>
> The value of is one of the Boolean literals or . Absence of the attribute means .

6.3 Open Entity Type

An entity type MAY indicate that it is open and allows clients to add properties dynamically to instances of the type by specifying uniquely named property values in the payload used to insert or update an instance of the type.

An entity type derived from an open entity type MUST indicate that it is also open.

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

> **undefined.1**
>
> The value of is one of the Boolean literals or . Absence of the attribute means .

6.4 Media Entity Type

An entity type that does not specify a base type MAY indicate that it is a media entity type. *Media entities* are entities that represent a media stream, such as a photo. Use a media entity if the out-of-band stream is the main topic of interest and the media entity is just additional structured information attached to the stream. Use a normal entity with one or more properties of type if the structured data of the entity is the main topic of interest and the stream data is just additional

information attached to the structured data. For more information on media entities see **[OData-Protocol]**.

An entity type derived from a media entity type MUST indicate that it is also a media entity type.

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

> **Attribute _____**
>
> The value of _____ is one of the Boolean literals _____ or _____. Absence of the attribute means _____.

6.5 Key

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

In order to be specified as the type of an entity set or a collection-valued containment navigation property, the entity type MUST either specify a key or inherit its key from its base type.

In OData 4.01 responses entity types used for singletons or single-valued navigation properties do not require a key. In OData 4.0 responses entity types used for singletons or single-valued navigation properties MUST have a key defined.

An entity type (whether or not it is marked as abstract) MAY define a key only if it doesn't inherit one.

An entity type's key refers to the set of properties whose values uniquely identify an instance of the entity type within an entity set. The key MUST consist of at least one property.

Key properties MUST NOT be nullable and MUST be typed with an enumeration type, one of the following primitive types, or a type definition based on one of these primitive types:

-
-
-
-
-
-
-
-
-
-
-
-
-

Key property values MAY be language-dependent, but their values MUST be unique across all languages and the entity ids (defined in **[OData-Protocol]**) MUST be language independent.

A key property MUST be a non-nullable primitive property of the entity type itself, including non-nullable primitive properties of non-nullable single-valued complex properties, recursively.

In OData 4.01 the key properties of a directly related entity type MAY also be part of the key if the navigation property is single-valued and not nullable. This includes navigation properties of non-nullable single-valued complex properties (recursively) of the entity type. If a key property of a related entity type is part of the key, all key properties of the related entity type MUST also be part of the key.

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

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

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

> **Element**
>
> The element MUST contain at least one element.

> **Element**
>
> The element MUST contain the attribute and MAY contain the attribute.

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

> **Attribute**
>
> The value of is a simple identifier.

Example 10: entity type with a simple key

```
<EntityType Name="Category">
  <Key>
    <PropertyRef Name="ID" />
  </Key>
  <Property Name="ID" Type="Edm.Int32" Nullable="false" />
  <Property Name="Name" Type="Edm.String" />
</EntityType>
```

Example 11: entity type with a simple key referencing a property of a complex type


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

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

Example 12: entity type with a composite key

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

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

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

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

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


7 Structural Property

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

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

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

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

Names are case-sensitive, but service authors SHOULD NOT choose names that differ only in case.

> **Element**
>
> The element MUST contain the and the attribute, and it MAY contain the facet attributes , , , , , , and .
>
> It MAY contain elements.
>
> **Attribute**
>
> The value of is the property's name.

Example 15: complex type with two properties

```
<ComplexType Name="Measurement">
  <Property Name="Dimension" Type="Edm.String" Nullable="false" MaxLength="50"
            DefaultValue="Unspecified" />
  <Property Name="Length" Type="Edm.Decimal" Nullable="false" Precision="18"
            Scale="2" />
</ComplexType>
```

7.1 Type

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

A collection-valued property MAY be annotated with the term, defined in **[OData-VocCore]**, to specify that it supports a stable ordering.

A collection-valued property MAY be annotated with the term, defined in **[OData-VocCore]**, to specify that it supports inserting items into a specific ordinal position.

> **Attribute**
>
> For single-valued properties the value of is the qualified name of the property's type.
>
> For collection-valued properties the value of is the character sequence followed by the qualified name of the property's item type, followed by a closing parenthesis .

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

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

7.2 Type Facets

Facets modify or constrain the acceptable values of a property.

For single-valued properties the facets apply to the value of the property. For collection-valued properties the facets apply to the items in the collection.

7.2.1 Nullable

A Boolean value specifying whether the property can have the value .

> **Attribute**
>
> The value of is one of the Boolean literals or .
>
> For single-valued properties the value means that the property allows the value.
>
> For collection-valued properties the property value will always be a collection that MAY be empty. In this case the attribute applies to items of the collection and specifies whether the collection MAY contain values.
>
> If no value is specified for a single-valued property, the attribute defaults to .
>
> In OData 4.01 responses a collection-valued property MUST specify a value for the attribute.
>
> If no value is specified for a collection-valued property, the client cannot assume any default value. Clients SHOULD be prepared for this situation even in OData 4.01 responses.

7.2.2 MaxLength

A positive integer value specifying the maximum length of a binary, stream or string value. For binary or stream values this is the octet length of the binary data, for string values it is the character length (number of code points for Unicode).

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

> **Attribute**
>
> The value of is a positive integer or the symbolic value as a shorthand for the maximum length supported for the type by the service.
>
> Note: the symbolic value is only allowed in OData 4.0 responses; it is deprecated in OData 4.01. While clients MUST be prepared for this symbolic value, OData 4.01 and greater services MUST NOT return the symbolic value and MAY instead specify the concrete maximum length supported for the type by the service or omit the attribute entirely.

7.2.3 Precision

For a decimal value: the maximum number of significant decimal digits of the property's value; it MUST be a positive integer.

For a temporal value (datetime-with-timezone-offset, duration, or time-of-day): the number of decimal places allowed in the seconds portion of the value; it MUST be a non-negative integer between zero and twelve.

Note: service authors SHOULD be aware that some clients are unable to support a precision greater than 28 for decimal properties and 7 for temporal properties. Client developers MUST be aware of the potential for data loss when round-tripping values of greater precision. Updating via and exclusively specifying modified properties will reduce the risk for unintended data loss.

Note: duration properties supporting a granularity less than seconds (e.g. minutes, hours, days) can be annotated with term , see **[OData-VocMeasures]**.

> **Attribute**
>
> The value of is a number.
>
> If not specified for a decimal property, the decimal property has arbitrary precision.
>
> If not specified for a temporal property, the temporal property has a precision of zero.

Example 17: facet applied to the type

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

7.2.4 Scale

A non-negative integer value specifying the maximum number of digits allowed to the right of the decimal point, or one of the symbolic values or .

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

The value means that the number of digits to the right of the decimal point can vary from zero to the value of the facet.

An integer value means that the number of digits to the right of the decimal point may vary from zero to the value of the facet, and the number of digits to the left of the decimal point may vary from one to the value of the facet minus the value of the facet. If is equal to , a single zero MUST precede the decimal point.

The value of MUST be less than or equal to the value of _____.

Note: if the underlying data store allows negative scale, services may use a _____ with the absolute value of the negative scale added to the actual number of significant decimal digits, and client-provided values may have to be rounded before being stored.

> **Attribute**
>
> The value of is a number or one of the symbolic values or .
>
> Services SHOULD use lower-case values; clients SHOULD accept values in a case-insensitive manner.
>
> If not specified, the facet defaults to zero.

Example 18: _____ and . Allowed values: 1.23, 0.23, 3.14 and 0.7, not allowed values: 123, 12.3

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

Example 19: equals . Allowed values: 0.23, 0.7, not allowed values: 1.23, 1.2

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

Example 20: and a variable . Allowed values: 0.123, 1.23, 0.23, 0.7, 123 and 12.3, not allowed values: 12.34, 1234 and 123.4 due to the limited precision.

```
<Property Name="Amount3v" Type="Edm.Decimal" Precision="3" Scale="variable" />
```

Example 21: and a floating . Allowed values: -1.234567e3, 1e-101, 9.999999e96, not allowed values: 1e-102 and 1e97 due to the limited precision.

```
<Property Name="Amount7f" Type="Edm.Decimal" Precision="7" Scale="floating" />
```

7.2.5 Unicode

For a string property the facet indicates whether the property might contain and accept string values with Unicode characters (code points) beyond the ASCII character set. The value indicates that the property will only contain and accept string values with characters limited to the ASCII character set.

If no value is specified, the facet defaults to .

> **Attribute**
>
> The value of is one of the Boolean literals or . Absence of the attribute means .

7.2.6 SRID

For a geometry or geography property the facet identifies which spatial reference system is applied to values of the property on type instances.

The value of the facet MUST be a non-negative integer or the special value . If no value is specified, the facet defaults to for types or for types.

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

> **Attribute**
>
> The value of is a number or the symbolic value .

7.2.7 Default Value

A primitive or enumeration property MAY define a default value that is used if the property is not explicitly represented in an annotation or the body of a request or response.

If no value is specified, the client SHOULD NOT assume a default value.

> **Attribute**
>
> Default values of type MUST be represented according to the XML escaping rules for character data in attribute values. Values of other primitive types MUST be represented according to the appropriate alternative in the rule defined in **[OData-ABNF]**, i.e. as , as etc.

8 Navigation Property

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

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

Names are case-sensitive, but service authors SHOULD NOT choose names that differ only in case.

Element

The element MUST contain the and attributes, and it MAY contain the attributes , , and .

It MAY contain child elements and at most one child element .

It MAY contain elements.

Attribute

The value of is the navigation property's name.

Example 22: the Product entity type has a navigation property to a Category, which has a navigation link back to one or more products

```
<EntityType Name="Product">
  ...
  <NavigationProperty Name="Category" Type="self.Category" Nullable="false"
                      Partner="Products" />
  <NavigationProperty Name="Supplier" Type="self.Supplier" />
</EntityType>

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

8.1 Navigation Property Type

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

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

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

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

A collection-valued navigation property MAY be annotated with the _____ term, defined in **[OData-VocCore]**, to specify that it supports a stable ordering.

A collection-valued navigation property MAY be annotated with the _____ term, defined in **[OData-VocCore]**, to specify that it supports inserting items into a specific ordinal position.

> **Attribute _____**
>
> For single-valued navigation properties the value of is the qualified name of the navigation property's type.
>
> For collection-valued navigation properties the value of is the character sequence followed by the qualified name of the navigation property's item type, followed by a closing parenthesis .

8.2 Nullable Navigation Property

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

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

> **Attribute _____**
>
> The value of is one of the Boolean literals or . Absence of the attribute means .

8.3 Partner Navigation Property

A navigation property of an entity type MAY specify a partner navigation property. Navigation properties of complex types MUST NOT specify a partner.

If specified, the partner navigation property is identified by a path relative to the entity type specified as the type of the navigation property. This path MUST lead to a navigation property defined on that type or a derived type. The path MAY traverse complex types, including derived complex types, but MUST NOT traverse any navigation properties. The type of the partner navigation property MUST be the declaring entity type of the current navigation property or one of its parent entity types.

If the partner navigation property is single-valued, it MUST lead back to the source entity from all related entities. If the partner navigation property is collection-valued, the source entity MUST be part of that collection.

If no partner navigation property is specified, no assumptions can be made as to whether one of the navigation properties on the target type will lead back to the source entity.

If a partner navigation property is specified, this partner navigation property MUST either specify the current navigation property as its partner to define a bi-directional relationship or it MUST NOT specify a partner navigation property. The latter can occur if the partner navigation property is defined on a complex type, or if the current navigation property is defined on a type derived from the type of the partner navigation property.

> **Attribute**
>
> The value of is the path to the of the partner navigation property.

8.4 Containment Navigation Property

A navigation property MAY indicate that instances of its declaring structured type contain the targets of the navigation property, in which case the navigation property is called a *containment navigation property*.

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

Instances of the structured type that declares the navigation property, either directly or indirectly via a property of complex type, contain the entities referenced by the containment navigation property. The canonical URL for contained entities is the canonical URL of the containing instance, followed by the path segment of the navigation property and the key of the contained entity, see **[OData-URL]**.

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

For items of an ordered collection of complex types (those annotated with the term defined in **[OData-VocCore]** the canonical URL of the item is the canonical URL of the collection appended with a segment containing the zero-based ordinal of the item. Items within in an unordered collection of complex types do not have a canonical URL. Services that support unordered collections of complex types declaring a containment navigation property, either directly or indirectly via a property of complex type, MUST specify the URL for the navigation link within a payload representing that item, according to format-specific rules.

OData 4.0 responses MUST NOT specify a complex type declaring a containment navigation property as the type of a collection-valued property.

An entity cannot be referenced by more than one containment relationship, and cannot both belong to an entity set declared within the entity container and be referenced by a containment relationship.

Containment navigation properties MUST NOT be specified as the last path segment in the path of a navigation property binding.

When a containment navigation property navigates between entity types in the same inheritance hierarchy, the containment is called *recursive*.

Containment navigation properties MAY specify a partner navigation property. If the containment is recursive, the relationship defines a tree, thus the partner navigation property MUST be nullable (for the root of the tree) and single-valued (for the parent of a non-root entity). If the containment is not recursive, the partner navigation property MUST NOT be nullable.

An entity type inheritance chain MUST NOT contain more than one navigation property with a partner navigation property that is a containment navigation property.

Note: without a partner navigation property, there is no reliable way for a client to determine which entity contains a given contained entity. This may lead to problems for clients if the contained entity can also be reached via a non-containment navigation path.

> **Attribute**
>
> The value of is one of the Boolean literals or . Absence of the attribute means .

8.5 Referential Constraint

A single-valued navigation property MAY define one or more referential constraints. A referential constraint asserts that the *dependent property* (the property defined on the structured type declaring the navigation property) MUST have the same value as the *principal property* (the referenced property declared on the entity type that is the target of the navigation).

The type of the dependent property MUST match the type of the principal property, or both types MUST be complex types.

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

If the principle property's value is a complex type instance, then the dependent property's value must be a complex type instance with the same properties, each with the same values.

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

> **Element**
>
> The element MUST contain the attributes and .
>
> It MAY contain elements.
>
> **Attribute**
>
> The attribute specifies the property that takes part in the referential constraint on the dependent structured type. Its value MUST be a path expression resolving to a property of the dependent structured type itself or to a property of a complex property (recursively) of the dependent structured type. The names of the properties in the path are joined together by

forward slashes. The path is relative to the dependent structured type declaring the navigation property.

Attribute

The attribute specifies the corresponding property of the principal entity type. Its value MUST be a path expression resolving to a property of the principal entity type itself or to a property of a complex property (recursively) of the principal entity type that MUST have the same type as the property of the dependent entity type. The path is relative to the entity type that is the target of the navigation property.

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

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

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

8.6 On-Delete Action

A navigation property MAY define an on-delete action that describes the action the service will take on related entities when the entity on which the navigation property is defined is deleted.

The action can have one of the following values:

- , meaning the related entities will be deleted if the source entity is deleted,
- , meaning a request on a source entity with related entities will fail,
- , meaning all properties of related entities that are tied to properties of the source entity via a referential constraint and that do not participate in other referential constraints will be set to null,
- , meaning all properties of related entities that are tied to properties of the source entity via a referential constraint and that do not participate in other referential constraints will be set to their default value.

If no on-delete action is specified, the action taken by the service is not predictable by the client and could vary per entity.

> **Element**
>
> The element MUST contain the attribute.
>
> It MAY contain elements.
>
> **Attribute**
>
> The value of is one of the values , , , or .

Example 24: deletion of a category implies deletion of the related products in that category

```
<EntityType Name="Category">
  ...
  <NavigationProperty Name="Products" Type="Collection(self.Product)">
    <OnDelete Action="Cascade">
      <Annotation Term="Core.Description"
                  String="Delete all products in this category" />
    </OnDelete
  </NavigationProperty>
</EntityType>
```

9 Complex Type

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

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

A complex type can define two types of properties. A structural property is a named reference to a primitive, complex, or enumeration type, or a collection of primitive, complex, or enumeration types. A navigation property is a named reference to an entity type or a collection of entity types.

All properties MUST have a unique name within a complex type. Properties MUST NOT have the same name as the declaring complex type. They MAY have the same name as one of the direct or indirect base types or derived types.

Element

The element MUST contain the attribute, and it MAY contain the , , and attributes.

It MAY contain and elements describing the properties of the complex type.

It MAY contain elements.

Attribute

The value of is the complex type's name.

Example 25: a complex type used by two entity types

```
<ComplexType Name="Dimensions">
  <Property Name="Height" Nullable="false" Type="Edm.Decimal" />
  <Property Name="Weight" Nullable="false" Type="Edm.Decimal" />
  <Property Name="Length" Nullable="false" Type="Edm.Decimal" />
</ComplexType>

<EntityType Name="Product">
  ...
  <Property Name="ProductDimensions" Type="self.Dimensions" />
  <Property Name="ShippingDimensions" Type="self.Dimensions" />
</EntityType>

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

9.1 Derived Complex Type

A complex type can inherit from another complex type by specifying it as its base type.

A complex type inherits the structural and navigation properties of its base type.

A complex type MUST NOT introduce an inheritance cycle by specifying a base type.

The rules for annotations of derived complex types are described in section 14.2.

> **Attribute**
>
> The value of is the qualified name of the base type.

9.2 Abstract Complex Type

A complex type MAY indicate that it is abstract and cannot have instances.

> **Attribute**
>
> The value of is one of the Boolean literals or . Absence of the attribute means .

9.3 Open Complex Type

A complex type MAY indicate that it is open and allows clients to add properties dynamically to instances of the type by specifying uniquely named property values in the payload used to insert or update an instance of the type.

A complex type derived from an open complex type MUST indicate that it is also open.

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

> **Attribute**
>
> The value of is one of the Boolean literals or . Absence of the attribute means .

10 Enumeration Type

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

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

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

Enumeration types marked as flags allow values that consist of more than one enumeration member at a time.

> **Element**
>
> The element MUST contain the Name attribute, and it MAY contain the and attributes.
>
> It MUST contain one or more elements defining the members of the enumeration type.
>
> It MAY contain elements.
>
> **Attribute**
>
> The value of is the enumeration type's name.

Example 26: a simple flags-enabled enumeration

```
<EnumType Name="FileAccess" UnderlyingType="Edm.Int32" IsFlags="true">
  <Member Name="Read"   Value="1" />
  <Member Name="Write"  Value="2" />
  <Member Name="Create" Value="4" />
  <Member Name="Delete" Value="8" />
</EnumType>
```

10.1 Underlying Integer Type

An enumeration type MAY specify one of , , , , or as its underlying type.

If not explicitly specified, is used as the underlying type.

> **Attribute**
>
> The value of is the qualified name of the underlying type.

10.2 Flags Enumeration Type

An enumeration type MAY indicate that the enumeration type allows multiple members to be selected simultaneously.

If not explicitly specified, only one enumeration type member MAY be selected simultaneously.

> **Attribute**
>
> The value of is one of the Boolean literals or . Absence of the attribute means .

Example 27: pattern values can be combined, and some combined values have explicit names

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

10.3 Enumeration Type Member

Enumeration type values consist of discrete members.

Each member is identified by its name, a simple identifier that MUST be unique within the enumeration type. Names are case-sensitive, but service authors SHOULD NOT choose names that differ only in case.

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

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

Enumeration members are sorted by their numeric value.

For flag enumeration types the combined numeric value of simultaneously selected members is the bitwise OR of the discrete numeric member values.

> **Element**
>
> The element MUST contain the attribute and it MAY contain the attribute.

It MAY contain _____ elements.

Attribute _____

The value of is the enumeration member's name.

Attribute _____

If the _____ attribute has a value of , either all members MUST specify an integer value for the attribute, or all members MUST NOT specify a value for the attribute. If no values are specified, the members are assigned consecutive integer values in the order of their appearance, starting with zero for the first member. Client libraries MUST preserve elements in document order.

If the _____ attribute has a value of , a non-negative integer value MUST be specified for the attribute. A combined value is equivalent to the bitwise OR of the discrete values.

Example 28: has a value of , a value of 1, and a value of 2.

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

11 Type Definition

A type definition defines a specialization of one of the primitive types or of the built-in abstract type _____.

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

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

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

> **Element** _____
>
> The element MUST contain the and _____ attributes.
>
> It MAY contain _____ elements.
>
> **Attribute** _____
>
> The value of is the type definition's name.

Example 29:

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

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

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

11.1 Underlying Primitive Type

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

> **Attribute** _____
>
> The value of is the qualified name of the underlying type.

The type definition MAY specify facets applicable to the underlying type. Possible facets are: _____, _____, _____, ____, or ____.

Additional facets appropriate for the underlying type MAY be specified when the type definition is used but the facets specified in the type definition MUST NOT be re-specified.

For a type definition with underlying type _____ no facets are applicable, neither in the definition itself nor when the type definition is used, and these should be ignored by the client.

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

12 Action and Function

12.1 Action

Actions are service-defined operations that MAY have observable side effects and MAY return a single instance or a collection of instances of any type.

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

Actions cannot be composed with additional path segments.

An action MAY specify a return type that MUST be a primitive, entity or complex type, or a collection of primitive, entity or complex types in scope.

An action MAY define parameters used during the execution of the action.

12.2 Action Overloads

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

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

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

> **Element**
>
> The element MUST contain the attribute and it MAY contain the and attributes.
>
> It MAY contain at most one element and MAY contain elements.
>
> It MAY contain elements.
>
> **Attribute**
>
> The value of is the action's name.

12.3 Function

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

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

Functions MAY be composable.

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

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

12.4 Function Overloads

Bound functions support overloading (multiple functions having the same name within the same schema) subject to the following rules:

- The combination of function name, binding parameter type, and unordered set of non-binding parameter names MUST be unique within a schema.
- The combination of function name, binding parameter type, and ordered set of parameter types MUST be unique within a schema.
- All bound functions with the same function name and binding parameter type within a schema MUST specify the same return type.

Unbound functions support overloading subject to the following rules:

- The combination of function name and unordered set of parameter names MUST be unique within a schema.
- The combination of function name and ordered set of parameter types MUST be unique within a schema.
- All unbound functions with the same function name within a schema MUST specify the same return type.

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

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

> **Element**
>
> The element MUST contain the attribute and it MAY contain the and attributes.
>
> It MUST contain one element, and it MAY contain elements.
>
> It MAY contain elements.
>
> **Attribute**
>
> The value of is the action's name.

12.5 Bound or Unbound Action or Function Overloads

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

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

Unbound actions are invoked from the entity container through an action import.

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

> **Attribute**
>
> The value of is one of the Boolean literals or . Absence of the attribute means .

12.6 Entity Set Path

Bound actions and functions that return an entity or a collection of entities MAY specify an entity set path if the entity set of the returned entities depends on the entity set of the binding parameter value.

The entity set path consists of a series of segments joined together with forward slashes.

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

A navigation segment names the simple identifier of the navigation property to be traversed. A type-cast segment names the qualified name of the entity type that should be returned from the type cast.

> **Attribute**
>
> The value of is the entity set path.

12.7 Composable Function

A function MAY indicate that it is composable. If not explicitly indicated, it is not composable.

A composable function can be invoked with additional path segments or key predicates appended to the resource path that identifies the composable function, and with system query options as appropriate for the type returned by the composable function.

> **Attribute**
>
> The value of is one of the Boolean literals or . Absence of the attribute means .

12.8 Return Type

The return type of an action or function overload MAY be any type in scope, or a collection of any type in scope.

The facets , , , , and can be used as appropriate to specify value restrictions of the return type, as well as the facet for 4.01 and greater payloads.

For a single-valued return type the facets apply to the returned value. For a collection-valued return type the facets apply to the items in the returned collection.

Element

The element MUST contain the attribute, and it MAY contain the attributes , , , , , and .

It MAY contain elements.

Attribute

For single-valued return types the value of is the qualified name of the return type.

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

Attribute

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

If the return type is a collection of entity types, the attribute has no meaning and MUST NOT be specified.

For other collection-valued return types the result will always be a collection that MAY be empty. In this case the attribute applies to items of the collection and specifies whether the collection MAY contain values.

For single-valued return types the value means that the action or function MAY return a single value. The value means that the action or function will never return a value and instead will fail with an error response if it cannot compute a result.

12.9 Parameter

An action or function overload MAY specify parameters.

A bound action or function overload MUST specify at least one parameter; the first parameter is its binding parameter. The order of parameters MUST NOT change unless the schema version changes.

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

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

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

For single-valued parameters the facets apply to the parameter value. If the parameter value is a collection, the facets apply to the items in the collection.

Element

The element MUST contain the and the attribute, and it MAY contain the attributes , , , , and .

It MAY contain elements.

Attribute

The value of is the parameter's name.

Attribute

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

For collection-valued parameters the value of is the character sequence followed by the qualified name of the parameter's type, followed by a closing parenthesis .

Attribute

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

The value means that the parameter accepts a value.

Example 30: a function returning the top-selling products for a given year. In this case the year must be specified as a parameter of the function with the element.

```
<Function Name="TopSellingProducts">
  <Parameter Name="Year" Type="Edm.Decimal" Precision="4" Scale="0" />
  <ReturnType Type="Collection(self.Product)" />
</Function>
```

13 Entity Container

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

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

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

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

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

Example 31: one entity set per entity type

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

Other entity models may expose multiple entity sets per type.

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

```
<EntitySet Name="StandardCustomers" EntityType="self.Customer">
  <NavigationPropertyBinding Path="Orders" Target="Orders" />
</EntitySet>
<EntitySet Name="PreferredCustomers" EntityType="self.Customer">
  <NavigationPropertyBinding Path="Orders" Target="Orders" />
</EntitySet>
<EntitySet Name="Orders" EntityType="self.Order" />
```

There are separate entity sets for standard customers and preferred customers, but only one entity set for orders. The entity sets for standard customers and preferred customers both have navigation property bindings to the orders entity set, but the orders entity set does not have a navigation property binding for the Customer navigation property, since it could lead to either set of customers.

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

A *singleton* allows addressing a single entity directly from the entity container without having to know its key, and without requiring an entity set.

A *function import* or an *action import* is used to expose a function or action defined in an entity model as a top level resource.

Element

The MUST contain one or more , , , or elements.

It MAY contain elements.

Attribute

The value of is the entity container's name.

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

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

13.1 Extending an Entity Container

An entity container MAY specify that it extends another entity container in scope. All children of the "base" entity container are added to the "extending" entity container.

If the "extending" entity container defines an entity set with the same name as defined in any of its "base" containers, then the entity set's type MUST specify an entity type derived from the entity type specified for the identically named entity set in the "base" container. The same holds for singletons. Action imports and function imports cannot be redefined, nor can the "extending" container define a child with the same name as a child of a different kind in a "base" container.

Note: services should not introduce cycles by extending entity containers. Clients should be prepared to process cycles introduced by extending entity containers.

Attribute

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

Example 34: the entity container will contain all child elements that it defines itself, plus all child elements of the entity container located in


```
<EntityContainer Name="Extending" Extends="Some.Other.Schema.Base">
  ...
</EntityContainer>
```

13.2 Entity Set

Entity sets are top-level collection-valued resources.

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

An entity set MUST specify a type that MUST be an entity type in scope.

An entity set MUST contain only instances of its specified entity type or its subtypes. The entity type MAY be abstract but MUST have a key defined.

An entity set MAY indicate whether it is included in the service document. If not explicitly indicated, it is included.

Entity sets that cannot be queried without specifying additional query options SHOULD NOT be included in the service document.

Element

The element MUST contain the attributes and , and it MAY contain the attribute.

It MAY contain elements.

It MAY contain elements.

Attribute

The value of is the entity set's name.

Attribute

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

Attribute

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

13.3 Singleton

Singletons are top-level single-valued resources.

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

A singleton MUST specify a type that MUST be an entity type in scope.

A singleton MUST reference an instance its entity type.

Element

The element MUST include the attributes and , and it MAY contain the attribute.

It MAY contain elements.

It MAY contain elements.

Attribute

The value of is the singleton's name.

Attribute

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

Attribute

The value of is one of the Boolean literals or .

If no value is specified, the attribute defaults to .

In OData 4.0 responses this attribute MUST NOT be specified.

13.4 Navigation Property Binding

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

An entity set or a singleton SHOULD specify a navigation property binding for each navigation property of its entity type, including navigation properties defined on complex typed properties or derived types.

If omitted, clients MUST assume that the target entity set or singleton can vary per related entity.

13.4.1 Navigation Property Path Binding

A navigation property binding MUST specify a path to a navigation property of the entity set's or singleton's declared entity type, or a navigation property reached through a chain of type casts, complex properties, or containment navigation properties. If the navigation property is defined on a subtype, the path MUST contain the qualified name of the subtype, followed by a forward slash, followed by the navigation property name. If the navigation property is defined on a complex type used in the definition of the entity set's entity type, the path MUST contain a forward-slash separated list of complex property names and qualified type names that describe the path leading to the navigation property.

The path can traverse one or more containment navigation properties, but the last navigation property segment MUST be a non-containment navigation property and there MUST NOT be any non-

containment navigation properties prior to the final navigation property segment.

If the path traverses collection-valued complex properties or collection-valued containment navigation properties, the binding applies to all items of these collections.

If the path contains a recursive sub-path (i.e. a path leading back to the same structured type, the binding applies recursively to any positive number of cycles through that sub-path.

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

The same navigation property path MUST NOT be specified in more than one navigation property binding; navigation property bindings are only used when all related entities are known to come from a single entity set. Note that it is possible to have navigation property bindings for paths that differ only in a type-cast segment, allowing to bind instances of different sub-types to different targets. If paths differ only in type-cast segments, the most specific path applies.

13.4.2 Binding Target

A navigation property binding MUST specify a target via a simple identifier or target path. It specifies the entity set, singleton, or containment navigation property that contains the related entities.

If the target is a simple identifier, it MUST resolve to an entity set or singleton defined in the same entity container.

If the target is a target path, it MUST resolve to an entity set, singleton, or direct or indirect containment navigation property of a singleton in scope. The path can traverse single-valued containment navigation properties or single-valued complex properties before ending in a containment navigation property, and there MUST NOT be any non-containment navigation properties prior to the final segment.

> **Element**
>
> The element MUST contain the attributes and .
>
> **Attribute**
>
> The value of is a path expression.
>
> **Attribute**
>
> The value of is a target path.

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

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

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


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

Example 37: binding on contained within

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

13.5 Action Import

Action imports sets are top-level resources that are never included in the service document.

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

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

If the imported action returns an entity or a collection of entities, a simple identifier or target path value MAY be specified to identify the entity set that contains the returned entities. If a simple identifier is specified, it MUST resolve to an entity set defined in the same entity container. If a target path is specified, it MUST resolve to an entity set in scope.

Element

The element MUST contain the attributes and , and it MAY contain the attribute.

It MAY contain elements.

Attribute

The value of is the action import's name.

Attribute

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

Attribute

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

13.6 Function Import

Function imports sets are top-level resources.

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

A function import MUST specify the name of an unbound function in scope. All unbound overloads of the imported function can be invoked from the entity container.

If the imported function returns an entity or a collection of entities, a simple identifier or target path value MAY be specified to identify the entity set that contains the returned entities. If a simple identifier is specified, it MUST resolve to an entity set defined in the same entity container. If a target path is specified, it MUST resolve to an entity set in scope.

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

Element

The element MUST contain the attributes and , and it MAY contain the attributes and .

Attribute

The value of is the function import's name.

Attribute

The value of is the qualified name of an unbound function.

Attribute

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

Attribute

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

14 Vocabulary and Annotation

Vocabularies and annotations provide the ability to annotate metadata as well as instance data, and define a powerful extensibility point for OData. An *annotation* applies a *term* to a model element and defines how to calculate a value for the applied term.

Metadata annotations are terms applied to model elements. Behaviors or constraints described by a metadata annotation must be consistent with the annotated model element. Such annotations define additional behaviors or constraints on the model element, such as a service, entity type, property, function, action, or parameter. For example, a metadata annotation may define ranges of valid values for a particular property. Metadata annotations are applied in CSDL documents describing or referencing an entity model.

Instance annotations are terms applied to a particular instance within an OData payload, such as described in **[OData-JSON]**. An instance annotation can be used to define additional information associated with a particular result, entity, property, or error. For example, whether a property is read-only for a particular instance. Where the same annotation is defined at both the metadata and instance level, the instance-level annotation overrides the annotation specified at the metadata level. Annotations that apply across instances should be specified as metadata annotations.

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

A term can be used to:

- Extend model elements and type instances with additional information.
- Map instances of annotated structured types to an interface defined by the term type; i.e. annotations allow viewing instances of a structured type as instances of a differently structured type specified by the applied term.

A service SHOULD NOT require a client to interpret annotations. Clients SHOULD ignore invalid or unknown terms and silently treat unexpected or invalid values (including invalid type, invalid literal expression, invalid targets, etc.) as an unknown value for the term. Unknown or invalid annotations should never result in an error, as long as the payload remains well-formed.

Example 38: the entity type is extended with a by a metadata annotation that binds the term to the value of the property . The entity type also includes an annotation that allows its instances to be viewed as instances of the type specified by the term

```
<EntityType Name="Product">
  <Key>
    <PropertyRef Name="ID" />
  </Key>
  <Property Name="ID" Nullable="false" Type="Edm.Int32" />
  <Property Name="Name" Type="Edm.String" />
  <Property Name="Description" Type="Edm.String" />
  ...
  <Annotation Term="UI.DisplayName" Path="Name" />
```

```
  <Annotation Term="SearchVocabulary.SearchResult">
    <Record>
      <PropertyValue Property="Title" Path="Name" />
      <PropertyValue Property="Abstract" Path="Description" />
      <PropertyValue Property="Url">
        <Apply Function="odata.concat">
          <String>Products(</String>
          <Path>ID</Path>
          <String>)</String>
        </Apply>
      </PropertyValue>
    </Record>
  </Annotation>
</EntityType>
```

14.1 Term

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

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

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

Element

The element MUST contain the attributes and . It MAY contain the attributes and .

It MAY specify values for the , [ ]{.apple-converted-space}, , , or facet attributes, as well as the facet attribute for 4.01 and greater payloads. These facets and their implications are described in section 7.2.

A element whose attribute specifies a primitive or enumeration type MAY define a value for the attribute.

It MAY contain elements.

Attribute

The value of is the term's name.

Attribute

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

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

Attribute

The value of this attribute determines the value of the term when applied in an without providing an expression.

> Default values of type MUST be represented according to the XML escaping rules for character data in attribute values. Values of other primitive types MUST be represented according to the appropriate alternative in the rule defined in **[OData-ABNF]**, i.e. as , as etc.
>
> If no value is specified, the attribute defaults to .

14.1.1 Specialized Term

A term MAY specialize another term in scope by specifying it as its base term.

When applying a specialized term, the base term MUST also be applied with the same qualifier, and so on until a term without a base term is reached.

> **Attribute**
>
> The value of is the qualified name of the base term.

14.1.2 Applicability

The applicability of a term MAY be restricted to a list of model elements. If no list is supplied, the term is not intended to be restricted in its application. The list of model elements MAY be extended in future versions of the vocabulary. As the intended usage may evolve over time, clients SHOULD be prepared for any term to be applied to any model element and SHOULD be prepared to handle unknown values within the list of model constructs. Applicability is expressed using the following symbolic values:

Symbolic Value	Model Element
	Action
	Action Import
	Annotation
	Application of a client-side function in an annotation
	Type Cast annotation expression
	Entity Set or collection-valued Property or Navigation Property
	Complex Type
	Entity Container
	Entity Set
	Entity Type
	Enumeration Type

Symbolic Value	Model Element
	Function
	Function Import
	Conditional annotation expression
	Reference to an Included Schema
	Type Check annotation expression
	Labeled Element expression
	Enumeration Member
	Navigation Property
	Null annotation expression
	On-Delete Action of a navigation property
	Action of Function Parameter
	Property of a structured type
	Property value of a Record annotation expression
	Record annotation expression
	Reference to another CSDL document
	Referential Constraint of a navigation property
	Return Type of an Action or Function
	Schema
	Singleton
	Term
	Type Definition
	UrlRef annotation expression

Attribute

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

*Example 39: the term can be applied to properties and terms that are of type (the type and the two terms are defined in **[OData-VocCore]**)*

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

14.2 Annotation

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

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

Element

The element MUST contain the attribute , and it MAY contain the attribute .

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

If no expression is specified for a term with a primitive type, the annotation evaluates to the default value of the term definition. If no expression is specified for a term with a complex type, the annotation evaluates to a complex instance with default values for its properties. If no expression is specified for a collection-valued term, the annotation evaluates to an empty collection.

An element can be used as a child of the model element it annotates, or as the child of an element that targets the model element to be annotated.

An element MAY contain elements that annotate the annotation.

Attribute

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

Example 40: term , once applied with a constant value, once with a path value

```
<Property Name="AmountInReportingCurrency" Type="Edm.Decimal">
  <Annotation Term="Measures.ISOCurrency" String="USD">
    <Annotation Term="Core.Description"
```



```
                String="The parent company's currency" />
  </Annotation>
</Property>
<Property Name="AmountInTransactionCurrency" Type="Edm.Decimal">
  <Annotation Term="Measures.ISOCurrency" Path="Currency" />
</Property>
<Property Name="Currency" Type="Edm.String" MaxLength="3" />
```

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

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

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

14.2.1 Qualifier

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

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

> **Attribute** _____
>
> Annotation elements that are children of an _____ element MUST NOT provide a value for the qualifier attribute if the parent _____ element provides a value for the qualifier attribute.

Example 41: annotation should only be applied to tablet devices

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

14.2.2 Target

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

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

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

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

- Action (single or all overloads)
- Action Import
- Complex Type
- Entity Container
- Entity Set
- Entity Type
- Enumeration Type
- Enumeration Type Member
- Function (single or all overloads)
- Function Import
- Navigation Property (via type, entity set, or singleton)
- Parameter of an action or function (single overloads or all overloads defining the parameter)
- Property (via type, entity set, or singleton)
- Return Type of an action or function (single or all overloads)
- Singleton
- Type Definition

These are the direct children of a schema with a unique name (i.e. except actions and functions whose overloads to not possess a natural identifier), and all direct children of an entity container.

External targeting is possible for actions, functions, their parameters, and their return type, either in a way that applies to all overloads of the action or function or all parameters of that name across all overloads, or in a way that identifies a single overload.

External targeting is also possible for properties and navigation properties of singletons or entities in a particular entity set. These annotations override annotations on the properties or navigation properties targeted via the declaring structured type.

The allowed path expressions are:

- qualified name of schema child
- qualified name of schema child followed by a forward slash and name of child element
- qualified name of structured type followed by zero or more property, navigation property, or type-cast segments, each segment starting with a forward slash
- qualified name of an entity container followed by a segment containing a singleton or entity set name and zero or more property, navigation property, or type-cast segments
- qualified name of an action followed by parentheses containing the qualified name of the binding parameter *type* of a bound action overload to identify that bound overload, or by empty

parentheses to identify the unbound overload

- qualified name of a function followed by parentheses containing the comma-separated list of qualified names of the parameter *types* of a bound or unbound function overload in the order of their definition in the function overload
- qualified name of an action or function, optionally followed by parentheses as described in the two previous bullet points to identify a single overload, followed by a forward slash and either a parameter name or
- qualified name of an entity container followed by a segment containing an action or function import name, optionally followed by a forward slash and either a parameter name or
- One of the preceding, followed by a forward slash, an at ( ), the qualified name of a term, and optionally a hash ( ) and the qualifier of an annotation

All qualified names used in a target path MUST be in scope.

Example 42: Target expressions

```
MySchema.MyEntityType
MySchema.MyEntityType/MyProperty
MySchema.MyEntityType/MyNavigationProperty
MySchema.MyComplexType
MySchema.MyComplexType/MyProperty
MySchema.MyComplexType/MyNavigationProperty
MySchema.MyEnumType
MySchema.MyEnumType/MyMember
MySchema.MyTypeDefinition
MySchema.MyTerm
MySchema.MyEntityContainer
MySchema.MyEntityContainer/MyEntitySet
MySchema.MyEntityContainer/MySingleton
MySchema.MyEntityContainer/MyActionImport
MySchema.MyEntityContainer/MyFunctionImport
MySchema.MyAction
MySchema.MyAction(MySchema.MyBindingType)
MySchema.MyAction()
MySchema.MyFunction
MySchema.MyFunction(MySchema.MyBindingParamType,First.NonBinding.ParamType)
MySchema.MyFunction(First.NonBinding.ParamType,Second.NonBinding.ParamType)
MySchema.MyFunction/MyParameter
MySchema.MyEntityContainer/MyEntitySet/MyProperty
MySchema.MyEntityContainer/MyEntitySet/MyNavigationProperty
MySchema.MyEntityContainer/MyEntitySet/MySchema.MyEntityType/MyProperty
MySchema.MyEntityContainer/MyEntitySet/MySchema.MyEntityType/MyNavProperty
MySchema.MyEntityContainer/MyEntitySet/MyComplexProperty/MyProperty
MySchema.MyEntityContainer/MyEntitySet/MyComplexProperty/MyNavigationProperty
MySchema.MyEntityContainer/MySingleton/MyComplexProperty/MyNavigationProperty
```

14.3 Constant Expression

Constant expressions allow assigning a constant value to an applied term.

14.3.1 Binary

Expression

The expression evaluates to a primitive binary value. A binary expression MUST be assigned a value conforming to the rule in **[OData-ABNF]**.

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

Example 43: base64url-encoded binary value (OData)

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

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

14.3.2 Boolean

Expression

The expression evaluates to a primitive Boolean value. A Boolean expression MUST be assigned a Boolean value.

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

Example 44:

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

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

14.3.3 Date

Expression

The expression evaluates to a primitive date value. A date expression MUST be assigned a value of type , see **[XML-Schema-2]**, section 3.3.9. The value MUST also conform to rule in **[OData-ABNF]**, i.e. it MUST NOT contain a time-zone offset.

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

Example 45:

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

<Annotation Term="org.example.vCard.birthDay">
```

```
  <Date>2000-01-01</Date>
</Annotation>
```

14.3.4 DateTimeOffset

Expression

The expression evaluates to a primitive datetimestamp value with a time-zone offset. A datetimestamp expression MUST be assigned a value of type , see **[XML-Schema-2]**, section 3.4.28. The value MUST also conform to rule in **[OData-ABNF]**, i.e. it MUST NOT contain an end-of-day fragment (24:00:00).

The datetimestamp expression MAY be provided using element notation or attribute notation

Example 46:

```
<Annotation Term="org.example.display.LastUpdated"

            DateTimeOffset="2000-01-01T16:00:00.000Z" />

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

14.3.5 Decimal

Expression

The expression evaluates to a primitive decimal value. A decimal expression MUST be assigned a value conforming to the rule in **[OData-ABNF]**.

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

Example 47: attribute notation

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

Example 48: element notation

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

14.3.6 Duration

Expression

The expression evaluates to a primitive duration value. A duration expression MUST be assigned a value of type , see **[XML-Schema-2]**, section 3.4.27.

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

Example 49:

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

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

14.3.7 Enumeration Member

Expression

The expression references a member of an enumeration type. An enumeration member expression MUST be assigned a value that consists of the qualified name of the enumeration type, followed by a forward slash and the name of the enumeration member. If the enumeration type specifies an attribute with value , the expression MAY also be assigned a whitespace-separated list of values. Each of these values MUST resolve to the name of a member of the enumeration type of the specified term.

The enumeration member expression MAY be provided using element notation or attribute notation.

Example 50: single value

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

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

Example 51: combined value for enumeration type

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

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

14.3.8 Floating-Point Number

Expression

The expression evaluates to a primitive floating point (or double) value. A float expression MUST be assigned a value conforming to the rule in **[OData-ABNF]**.

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

Example 52:

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

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

14.3.9 Guid

Expression

The expression evaluates to a primitive guid value. A guid expression MUST be assigned a value conforming to the rule in **[OData-ABNF]**.

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

Example 53:

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

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

14.3.10 Integer

Expression

The expression evaluates to a primitive integer value. An integer MUST be assigned a value conforming to the rule in **[OData-ABNF]**.

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

Example 54: attribute notation

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

Example 55: element notation

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

14.3.11 String

Expression

The expression evaluates to a primitive string value. A string expression MUST be assigned a value of the type , see **[XML-Schema-2]**, section 3.3.1.

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

Example 56:

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

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

14.3.12 Time of Day

Expression

The expression evaluates to a primitive time value. A time-of-day expression MUST be assigned a value conforming to the rule in **[OData-ABNF]**.

The time-of-day expression MAY be provided using element notation or attribute notation.

Example 57:

```
<Annotation Term="org.example.display.EndTime" TimeOfDay="21:45:00" />

<Annotation Term="org.example.display.EndTime">
  <TimeOfDay>21:45:00</TimeOfDay>
</Annotation>
```

14.4 Dynamic Expression

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

14.4.1 Path Expressions

Path expressions allow assigning a value to an applied term or term component. There are two kinds of path expressions:

- A *model path* is used within Annotation Path, Model Element Path, Navigation Property Path, and Property Path expressions to traverse the model of a service and resolves to the model element identified by the path. It allows assigning values to terms or term properties of the built-in types , , , and their base types and .
- An *instance path* is used within a Value Path expression to traverse a graph of type instances and resolves to the value identified by the path. It allows assigning values to terms or term properties of built-in types other than the types, or of any model-defined type.

14.4.1.1 Path Syntax

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

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

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

Example 58: absolute path to an entity set

```
/My.Schema.MyEntityContainer/MyEntitySet
```

Paths not starting with a forward slash are interpreted relative to the annotation target, following the rules specified in section "Path Evaluation".

Example 59: relative path to a property

```
Address/City
```

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

Example 60: type-cast segment

```
.../self.Manager/...
```

If a path segment starts with an at ( ) character, it represents a *term cast*. The at ( ) character MUST be followed by a qualified name that MAY be followed by a hash ( ) character and a simple identifier. The qualified name preceding the hash character MUST resolve to a term that is in scope, the simple identifier following the hash sign is interpreted as a qualifier for the term. If the model element or instance identified by the preceding path part has not been annotated with that term (and if present, with that qualifier), the term cast evaluates to the null value. Four special terms are implicitly "annotated" for media entities and stream properties:

-
-
-
-

Example 61: term-cast segments

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

If a path segment is a simple identifier, it MUST be the name of a child model element of the model element identified by the preceding path part, or a structural or navigation property of the instance identified by the preceding path part. A sequence of navigation segments can traverse multiple CSDL documents. The document containing the path expression only needs to reference the next traversed document to bring the navigation target type into scope, and each traversed document in turn needs to reference only its next document.

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

Example 62: property segments in model path

```
.../Orders/Items/Product/...
```


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

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

Example 63: property segments in instance path

```
.../Addresses/Street
```

```
.../Addresses/$count
```

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

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

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

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

Example 65: instance path with entity set and key predicate

```
/self.container/SettingsCollection('FeatureXxx')/IsAvailable
```

```
/self.container/Products(ID=ProductID)/Name
```

An instance path MAY contain an index segment immediately following a path segment representing an ordered collection-valued structural property. The index is zero-based and MUST be an integer literal. Negative integers count from the end of the collection, with -1 representing the last item in the collection. Remaining path segments are evaluated relative to the identified item of the collection.

Example 66: instance path with collection-valued structural property and index segment

```
Addresses/1
```



```
Addresses/-1/Street
```

14.4.1.2 Path Evaluation

Annotations MAY be embedded within their target, or specified separately, e.g. as part of a different schema, and specify a path to their target model element. The latter situation is referred to as *targeting* in the remainder of this section.

For annotations embedded within or targeting an entity container, the path is evaluated starting at the entity container, i.e. an empty path resolves to the entity container, and non-empty paths MUST start with a segment identifying a container child (entity set, function import, action import, or singleton). The subsequent segments follow the rules for paths targeting the corresponding child element.

For annotations embedded within or targeting an entity set or a singleton, the path is evaluated starting at the entity set or singleton, i.e. an empty path resolves to the entity set or singleton, and non-empty paths MUST follow the rules for annotations targeting the declared entity type of the entity set or singleton.

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

For annotations embedded within a structural or navigation property of an entity type or complex type, the path is evaluated starting at the directly enclosing type. This allows e.g. specifying the value of an annotation on one property to be calculated from values of other properties of the same type. An empty path resolves to the enclosing type, and non-empty paths MUST follow the rules for annotations targeting the directly enclosing type.

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

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

14.4.1.3 Annotation Path

The annotation path expression provides a value for terms or term properties that specify the built-in types . Its argument is a model path with the following restriction:

- A non-null path MUST resolve to an annotation.

A term or term property of type can be annotated with term (see **[OData-VocValidation]**) if its intended value is an annotation path that ends in a term cast with one of the listed terms.

The value of the annotation path expression is the path itself, not the value of the annotation identified by the path. This is useful for terms that reuse or refer to other terms.

> **Expression**
>
> The expression MAY be provided using element notation or attribute notation.

Example 67:

```
<Annotation Term="UI.ReferenceFacet"
            AnnotationPath="Product/Supplier/@UI.LineItem" />

<Annotation Term="UI.CollectionFacet" Qualifier="Contacts">
  <Collection>
    <AnnotationPath>Supplier/@Communication.Contact</AnnotationPath>
    <AnnotationPath>Customer/@Communication.Contact</AnnotationPath>
  </Collection>
</Annotation>
```

14.4.1.4 Model Element Path

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

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

> **Expression**
>
> The expression MAY be provided using element notation or attribute notation.

Example 68:

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

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

14.4.1.5 Navigation Property Path

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

- A non-null path MUST resolve to a model element whose type is an entity type, or a collection of entity types, e.g. a navigation property.

The value of the navigation property path expression is the path itself, not the entitiy or collection of entities identified by the path.

> **Expression**
>
> The expression MAY be provided using element notation or attribute notation.

Example 69:

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

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

14.4.1.6 Property Path

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

- A non-null path MUST resolve to a model element whose type is a primitive or complex type, an enumeration type, a type definition, or a collection of one of these types.

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

> **Expression**
>
> The MAY be provided using either element notation or attribute notation.

Example 70:

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

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

14.4.1.7 Value Path

The value path expression allows assigning a value by traversing an object graph. It can be used in annotations that target entity containers, entity sets, entity types, complex types, navigation properties of structured types, and structural properties of structured types. Its argument is an instance path.

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

> **Expression**
>
> The expression MAY be provided using element notation or attribute notation.

Example 71:

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

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

14.4.2 Comparison and Logical Operators

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

Operator	Description
Logical Operators	
	Logical and
	Logical or
	Logical negation
Comparison Operators	
	Equal
	Not equal
	Greater than
	Greater than or equal
	Less than
	Less than or equal
	Has enumeration flag(s) set

Operator	Description
	Is in collection

The and operators require two operand expressions that evaluate to Boolean values. The operator requires a single operand expression that evaluates to a Boolean value. For details on null handling for comparison operators see **[OData-URL]**.

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

> ### Expressions and
>
> The and logical expressions are represented as elements and that MUST contain two annotation expressions.
>
> It MAY contain elements.
>
> ### Expression
>
> Negation expressions are represented as an element that MUST contain a single annotation expression.
>
> It MAY contain elements.
>
> ### Expressions , , , , , , , and
>
> All comparison expressions are represented as an element that MUST contain two annotation expressions.
>
> They MAY contain elements.

Example 72:

```
<And>
  <Path>IsMale</Path>
  <Path>IsMarried</Path>
</And>
<Or>
  <Path>IsMale</Path>
  <Path>IsMarried</Path>
</Or>
<Not>
  <Path>IsMale</Path>
</Not>
<Eq>
  <Null />
  <Path>IsMale</Path>
</Eq>
<Ne>
```

```
  <Null />
  <Path>IsMale</Path>
</Ne>
<Gt>
  <Path>Price</Path>
  <Int>20</Int>
</Gt>
<Ge>
  <Path>Price</Path>
  <Int>10</Int>
</Ge>
<Lt>
  <Path>Price</Path>
  <Int>20</Int>
</Lt>
<Le>
  <Path>Price</Path>
  <Int>100</Int>
</Le>
<Has>
  <Path>Fabric</Path>
  <EnumMember>org.example.Pattern/Red</EnumMember>
</Has>
<In>
  <Path>Size</Path>
  <Collection>
    <String>XS</String>
    <String>S</String>
  </Collection>
</In>
```

14.4.3 Arithmetic Operators

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

Operator	Description
	Addition
	Subtraction
	Negation
	Multiplication
	Division (with integer result for integer operands)
	Division (with fractional result also for integer operands)

Operator	Description
	Modulo

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

> **Expression**
>
> Negation expressions are represented as an element that MUST contain a single annotation expression.
>
> It MAY contain elements.
>
> **Expressions , , , , , and**
>
> These arithmetic expressions are represented as an element that MUST contain two annotation expressions.
>
> They MAY contain elements.

Example 73:

```
<Add>
  <Path>StartDate</Path>
  <Path>Duration</Path>
</Add>
<Sub>
  <Path>Revenue</Path>
  <Path>Cost</Path>
</Sub>
<Neg>
  <Path>Height</Path>
</Neg>
<Mul>
  <Path>NetPrice</Path>
  <Path>TaxRate</Path>
</Mul>
<Div>
  <Path>Quantity</Path>
  <Path>QuantityPerParcel</Path>
</Div>
<DivBy>
  <Path>Quantity</Path>
  <Path>QuantityPerParcel</Path>
</DivBy>
<Mod>
  <Path>Quantity</Path>
  <Path>QuantityPerParcel</Path>
</Mod>
```

14.4.4 Apply Client-Side Functions

The apply expression enables a value to be obtained by applying a client-side function. The apply expression MAY have operand expressions. The operand expressions are used as parameters to the client-side function.

> **Expression**
>
> The element MUST contain the attribute and MAY contain annotation expressions as operands for the applied function.
>
> It MAY contain more elements.
>
> **Attribute**
>
> The value of is the qualified name of the client-side function to apply.

OData defines the following functions. Services MAY support additional functions that MUST be qualified with a namespace other than . Function names qualified with are reserved for this specification and its future versions.

14.4.4.1 Canonical Functions

All canonical functions defined in **[OData-URL]** can be used as client-side functions, qualified with the namespace . The semantics of these client-side functions is identical to their counterpart function defined in **[OData-URL]**.

For example, the client-side function takes two or more expressions as arguments. Each argument MUST evaluate to a primitive or enumeration type. It returns a value of type that is the concatenation of the literal representations of the results of the argument expressions. Values of primitive types other than are represented according to the appropriate alternative in the rule of **[OData-ABNF]**, i.e. as , as etc.

Example 74:

```
<Annotation Term="org.example.display.DisplayName">
  <Apply Function="odata.concat">
    <String>Product: </String>
    <Path>ProductName</Path>
    <String> (</String>
    <Path>Available/Quantity</Path>
    <String> </String>
    <Path>Available/Unit</Path>
    <String> available)</String>
  </Apply>
</Annotation>
```

is of type , in complex type is of type , and in is of type enumeration, so the result of the expression is represented as the member name of the enumeration value.

14.4.4.2 Function

The client-side function takes two or more expressions as arguments and returns a value of type

The first argument MUST be of type and specifies a URI template according to **[RFC6570]**, the other arguments MUST be labeled element expressions. Each labeled element expression specifies the template parameter name as its name and evaluates to the template parameter value.

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

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

Lists of values are represented as labeled element expressions that evaluate to a collection of primitive values.

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

Example 75: assuming there are no special characters in values of the Name property of the Actor entity

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

14.4.4.3 Function

The client-side function takes two string expressions as arguments and returns a Boolean value

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

Example 76: all non-empty values not containing the letters , , or evaluate to

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

14.4.4.4 Function

The client-side function takes one argument of primitive type and returns the URL-encoded OData literal that can be used as a key value in OData URLs or in the query part of OData URLs.

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

Example 77:

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

14.4.5 Cast

The cast expression casts the value obtained from its single child expression to the specified type. The cast expression follows the same rules as the canonical function defined in **[OData-URL]**.

Expression

The element MUST contain the attribute and MUST contain exactly one expression.

It MAY contain elements.

Attribute

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

If the specified type is a primitive type or a collection of a primitive type, the facet attributes , , , , and MAY be specified if applicable to the specified primitive type. If the facet attributes are not specified, their values are considered unspecified.

Example 78:

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

14.4.6 Collection

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

Expression

The element contains zero or more child expressions.

Example 79:

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

14.4.7 If-Then-Else

The if-then-else expression enables a value to be obtained by evaluating a *condition expression*. It MUST contain exactly three child expressions. There is one exception to this rule: if and only if the if-then-else expression is an item of a collection expression, the third child expression MAY be omitted, reducing it to an if-then expression. This can be used to conditionally add an element to a collection.

The first child expression is the condition and MUST evaluate to a Boolean result, e.g. the comparison and logical operators can be used.

The second and third child expressions are evaluated conditionally. The result MUST be type compatible with the type expected by the surrounding expression.

If the first expression evaluates to , the second expression MUST be evaluated and its value MUST be returned as the result of the if-then-else expression. If the first expression evaluates to and a third child element is present, it MUST be evaluated and its value MUST be returned as the result of the if-then-else expression. If no third expression is present, nothing is added to the surrounding collection.

> **Expression**
>
> The element MUST contain two or three child expressions that MUST use element notation.
>
> It MAY contain elements.

Example 80: the condition is a value path expression referencing the Boolean property , whose value then determines the value of the expression (or so it was long ago)

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

14.4.8 Is-Of

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

Expression

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

Relative URLs are relative to the attribute, see **[XML-Base]**.

In element notation it MAY contain elements.

Example 81:

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

14.4.9 Labeled Element

The labeled element expression assigns a name to its single child expression. The value of the child expression can then be reused elsewhere with a labeled element reference expression.

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

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

Expression

The element MUST contain the Name attribute.

It MUST contain a child expression written either in attribute notation or element notation.

It MAY contain elements.

Attribute

The value of is the labeled element's name.

Example 82:

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

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


14.4.10 Labeled Element Reference

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

> **Expression**
>
> The element MUST contain the qualified name of a labeled element expression in its body.

Example 83:

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

14.4.11 Null

The null expression indicates the absence of a value. The null expression MAY be annotated.

> **Expression**
>
> The element MAY contain elements.

Example 84:

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

Example 85:

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

14.4.12 Record

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

A record expression MAY specify the structured type of its result, which MUST be an entity type or complex type in scope. If not explicitly specified, the type is derived from the expression's context.

A record expression contains zero or more property value expressions. For each single-valued structural or navigation property of the record expression's type that is neither nullable nor specifies a default value a property value expression MUST be provided. The only exception is if the record expression is the value of an annotation for a term that has a base term whose type is structured and directly or indirectly inherits from the type of its base term. In this case, property values that already have been specified in the annotation for the base term or its base term etc. need not be specified again.

For collection-valued properties the absence of a property value expression is equivalent to specifying an empty collection as its value.

Expression

The element MAY contain the attribute and MAY contain elements.

It MAY contain elements.

Attribute

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

Element

The element MUST contain the attribute, and it MUST contain exactly one expression that MAY be provided using either element notation or attribute notation.

It MAY contain elements.

Attribute

The value of is the name of a property of the type of the enclosing expression.

Example 86: this annotation "morphs" the entity type from example 8 into a structured type with two structural properties and and two navigation properties and . The first three properties simply rename properties of the annotated entity type, the fourth adds a calculated navigation property that is pointing to a different service

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

14.4.13 URL Reference

The URL reference expression enables a value to be obtained by sending a request.

The URL reference expression MUST contain exactly one expression of type . Its value is treated as a URL that MAY be relative or absolute; relative URLs are relative to the URL of the document containing the URL reference expression, or relative to a base URL specified in a format-specific way.

The response body of the request MUST be returned as the result of the URL reference expression. The result of the URL reference expression MUST be type compatible with the type expected by the surrounding expression.

> **Expression**
>
> The expression MAY be provided using element notation or attribute notation.
>
> Relative URLs are relative to the attribute, see **[XML-Base]**.
>
> In element notation it MAY contain elements.

Example 87:

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

<Annotation Term="Core.LongDescription">
  <UrlRef><String>http://host/wiki/HowToUse</String></UrlRef>
</Annotation>

<Annotation Term="Core.LongDescription" UrlRef="http://host/wiki/HowToUse" />
```

15 Identifier and Path Values

15.1 Namespace

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

15.2 Simple Identifier

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

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

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

15.3 Qualified Name

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

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

15.4 Target Path

Target paths are used to refer to other model elements.

The allowed path expressions are:

- The qualified name of an entity container, followed by a forward slash and the name of a container child element
- The target path of a container child followed by a forward slash and one or more forward-slash separated property, navigation property, or type-cast segments

Example 88: Target expressions

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

16 CSDL Examples

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

16.1 Products and Categories Example

Example 89:

```
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx"
           xmlns="http://docs.oasis-open.org/odata/ns/edm" Version="4.0">
  <edmx:Reference Uri="https://oasis-tcs.github.io/odata-
        vocabularies/vocabularies/Org.OData.Core.V1.xml">
    <edmx:Include Namespace="Org.OData.Core.V1" Alias="Core">
      <Annotation Term="Core.DefaultNamespace" />
    </edmx:Include>
  </edmx:Reference>
  <edmx:Reference Uri="https://oasis-tcs.github.io/odata-
        vocabularies/vocabularies/Org.OData.Measures.V1.xml">
    <edmx:Include Alias="Measures" Namespace="Org.OData.Measures.V1" />
  </edmx:Reference>
  <edmx:DataServices>
    <Schema Namespace="ODataDemo">
      <EntityType Name="Product" HasStream="true">
        <Key>
          <PropertyRef Name="ID" />
        </Key>
        <Property Name="ID" Type="Edm.Int32" Nullable="false" />
        <Property Name="Description" Type="Edm.String" >
          <Annotation Term="Core.IsLanguageDependent" />
        </Property>
        <Property Name="ReleaseDate" Type="Edm.Date" />
        <Property Name="DiscontinuedDate" Type="Edm.Date" />
        <Property Name="Rating" Type="Edm.Int32" />
        <Property Name="Price" Type="Edm.Decimal" Scale="variable">
          <Annotation Term="Measures.ISOCurrency" Path="Currency" />
        </Property>
        <Property Name="Currency" Type="Edm.String" MaxLength="3" />
        <NavigationProperty Name="Category" Type="ODataDemo.Category"
                            Nullable="false" Partner="Products" />
        <NavigationProperty Name="Supplier" Type="ODataDemo.Supplier"
                            Partner="Products" />
      </EntityType>
      <EntityType Name="Category">
        <Key>
         <PropertyRef Name="ID" />
        </Key>
        <Property Name="ID" Type="Edm.Int32" Nullable="false" />
        <Property Name="Name" Type="Edm.String" Nullable="false">
          <Annotation Term="Core.IsLanguageDependent" />
        </Property>
        <NavigationProperty Name="Products" Partner="Category"
```



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



```
            </Collection>
          </Annotation>
        </EntitySet>
        <Singleton Name="MainSupplier" Type="self.Supplier">
          <NavigationPropertyBinding Path="Products" Target="Products" />
          <Annotation Term="Core.Description" String="Primary Supplier" />
        </Singleton>
        <EntitySet Name="Countries" EntityType="ODataDemo.Country" />
        <FunctionImport Name="ProductsByRating" EntitySet="Products"
                        Function="ODataDemo.ProductsByRating" />
      </EntityContainer>
    </Schema>
  </edmx:DataServices>
</edmx:Edmx>
```

16.2 Annotations for Products and Categories Example

Example 90:

```
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx"
           Version="4.01">
  <edmx:Reference Uri="http://host/service/$metadata">
    <edmx:Include Namespace="ODataDemo" Alias="target" />
  </edmx:Reference>
  <edmx:Reference Uri="http://somewhere/Vocabulary/V1">
    <edmx:Include Alias="Vocabulary1" Namespace="Some.Vocabulary.V1" />
  </edmx:Reference>
  <edmx:DataServices>
    <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm"
            Namespace="External.Annotations">
      <Annotations Target="ODataDemo.Supplier">
        <Annotation Term="Vocabulary1.EMail">
          <Null />
        </Annotation>
        <Annotation Term="Vocabulary1.AccountID" Path="ID" />
        <Annotation Term="Vocabulary1.Title" String="Supplier Info" />
        <Annotation Term="Vocabulary1.DisplayName">
        <Apply Function="odata.concat">
            <Path>Name</Path>
            <String> in </String>
            <Path>Address/CountryName</Path>
          </Apply>
        </Annotation>
      </Annotations>
      <Annotations Target="ODataDemo.Product">
        <Annotation Term="Vocabulary1.Tags">
          <Collection>
            <String>MasterData</String>
          </Collection>
        </Annotation>
      </Annotations>
  </Schema>
```

```
  </edmx:DataServices>
</edmx:Edmx>
```

17 Conformance

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

In addition, conforming services MUST NOT return 4.01 CSDL constructs for requests made with .

Specifically, they

1. MUST NOT include properties in derived types that overwrite a property defined in the base type
2. MUST NOT include
3. MUST NOT use path syntax added with 4.01
4. MUST NOT use and
5. MUST NOT specify referential constraints to complex types and navigation properties
6. MUST NOT include a non-abstract entity type with no inherited or defined entity key
7. MUST NOT include the _____ annotation on included schemas
8. MUST NOT return the Unicode facet for terms, parameters, and return types
9. MUST NOT include collections of or
10. MUST NOT specify a key as a property of a related entity
11. SHOULD NOT include new/unknown values for the _____ attribute
12. MAY include new CSDL annotations

In addition, OData 4.01 services:

13. SHOULD NOT have identifiers within a uniqueness scope (e.g. a schema, a structural type, or an entity container) that differ only by case

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

Appendix A. References

This appendix contains the normative and informative references that are used in this document.

While any hyperlinks included in this appendix were valid at the time of publication, OASIS cannot guarantee their long-term validity.

A.1 Normative References

The following documents are referenced in such a way that some or all of their content constitutes requirements of this document.

[OData-v4.02]

- *OData Version 4.02*. Edited by Michael Pizzo, Ralf Handl, and Heiko Theißen. A multi-part Work Product that includes:
  - *OData Version 4.02 Part 1: Protocol*. Latest stage. https://docs.oasis-open.org/odata/odata/v4.02/odata-v4.02-part1-protocol.html
  - *OData Version 4.02 Part 2: URL Conventions*. Latest stage. https://docs.oasis-open.org/odata/odata/v4.02/odata-v4.02-part2-url-conventions.html

[ECMAScript]

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

[EPSG]

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

[OData-ABNF]

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

[OData-CSDL-Schema]

OData CSDL JSON Schema.
See link in "Related work" section on cover page.

[OData-CSDL-JSON]

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

[OData-CSDL-XML]

OData Common Schema Definition Language (CSDL) XML Representation Version 4.01.
See link in "Related work" section on cover page.

[OData-EDM]

OData EDM XML Schema.
See link in "Additional artifacts" section on cover page.

[OData-EDMX]

OData EDM XML Schema.
See link in "Additional artifacts" section on cover page.

[OData-JSON]

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

[OData-Protocol]

OData Version 4.01 Part 1: Protocol.
See link in "Related work" section on cover page.

[OData-URL]

OData Version 4.01 Part 2: URL Conventions.
See link in "Related work" section on cover page.

[OData-VocCore]

OData Vocabularies Version 4.0: Core Vocabulary.
See link in "Related work" section on cover page.

[OData-VocMeasures]

OData Vocabularies Version 4.0: Measures Vocabulary.
See link in "Related work" section on cover page.

[OData-VocValidation]

OData Vocabularies Version 4.0: Validation Vocabulary.
See link in "Related work" section on cover page.

[RFC2119]

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.
https://www.rfc-editor.org/info/rfc2119.

[RFC6570]

Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, “URI Template”, RFC 6570, March 2012.
http://tools.ietf.org/html/rfc6570.

[RFC7493]

Bray, T., Ed., "The I-JSON Message Format", RFC7493, March 2015.
https://tools.ietf.org/html/rfc7493.

[RFC8174]

Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017.
http://www.rfc-editor.org/info/rfc8174.

[RFC8259]

Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 8259, December 2017.
http://tools.ietf.org/html/rfc8259.

[XML-1.1]

Extensible Markup Language (XML) 1.1 (Second Edition). F. Yergeau, E. Maler, J. Cowan, T. Bray, C. M. Sperberg-McQueen, J. Paoli, Editors, W3C Recommendation, 16 August 2006. http://www.w3.org/TR/2006/REC-xml11-20060816. Latest version available at http://www.w3.org/TR/xml11/.

[XML-Base]

XML Base (Second Edition). J. Marsh, R. Tobin, Editors, W3C Recommendation, 28 January 2009. http://www.w3.org/TR/2009/REC-xmlbase-20090128/. Latest version available at http://www.w3.org/TR/xmlbase/.

[XML-Schema-1]

W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. D. Beech, M. Maloney, C. M. Sperberg-McQueen, H. S. Thompson, S. Gao, N. Mendelsohn, Editors, W3C Recommendation, 5 April 2012.
http://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/. Latest version available at http://www.w3.org/TR/xmlschema11-1/.

[XML-Schema-2]

W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes. D. Peterson, S. Gao, C. M. Sperberg-McQueen, H. S. Thompson, P. V. Biron, A. Malhotra, Editors, W3C Recommendation, 5 April 2012.
http://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/. Latest version available at http://www.w3.org/TR/xmlschema11-2/.

A.2 Informative References

[OpenUI5]

OpenUI5 Version 1.40.10 - OData V4 Metadata JSON Format.
https://openui5.hana.ondemand.com/1.40.10/#docs/guide/87aac894a40640f89920d7b2a414499b.html.

Appendix B. Table of XML Elements and Attributes

Appendix C. Acknowledgments

C.1 Special Thanks

The work of the OpenUI5 team on the OData V4 Metadata JSON Format, see **[OpenUI5]**, is gratefully acknowledged, especially the contributions of

- Thomas Chadzelek (SAP SE)
- Jens Ittel (SAP SE)
- Patric Ksinsik (SAP SE)

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

C.2 Participants

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

OpenC2 TC Members:

First Name	Last Name	Company
Philippe	Alman	Something Networks
Alex	Amirnovman	Company B
Kris	Anderman	Mini Micro
Darren	Anstman	Big Networks

Appendix D. Revision History

Revision	Date	Editor	Changes Made
Committee Specification Draft 01	2023-07-14	Michael Pizzo Ralf Handl Heiko Theißen	

Appendix E. Notices

Copyright © OASIS Open 2023. All Rights Reserved.---------- Michael Pizzo All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.
Ralf Handl
This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

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

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.---------- Michael Pizzo As stated in the OASIS IPR Policy, the following three paragraphs in brackets apply to OASIS Standards Final Deliverable documents (Committee Specification, OASIS Standard, or Approved Errata).
Ralf Handl
[OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Standards Final Deliverable, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this deliverable.]---------- Michael Pizzo [OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this OASIS Standards Final Deliverable by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this OASIS Standards Final Deliverable. OASIS may include such claims on its website, but disclaims any obligation to do so.]
Ralf Handl
[OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this

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

Michael Pizzo
Ralf Handl

Michael Pizzo
Ralf Handl

Michael Pizzo
Ralf Handl

