**Open Data Protocol**

**Introduction**

The Open Data Protocol (OData) enables the creation of HTTP-based data services, which allow resources identified using Uniform Resource Identifiers (URIs) and defined in an abstract data model, to be published and edited by Web clients using simple HTTP messages. OData is intended to be used to expose and access information from a variety of sources including, but not limited to, relational databases, file systems, content management systems, and traditional Web sites.

OData is made up of a group of specifications. This specification describes OData's core concepts and the following specifications define optional extensions to the foundations defined in this document:

|  |  |
| --- | --- |
| **Specification** | **Description** |
| [[OData:URI]](http://www.odata.org/developers/protocols/uri-conventions) | Conventions for constructing URIs to identify the resources and metadata exposed by an OData service. |
| [[OData:Terms]](http://www.odata.org/developers/protocols/terminology) | Glossary of terms used by OData |
| [[OData:Operations]](http://www.odata.org/developers/protocols/operations) | Defines the request types (retrieve, insert, update, delete, etc) and associated responses used by the OData protocol. An implementation can support some or all of the request types. |
| [[OData:Atom]](http://www.odata.org/developers/protocols/atom-format) | Defines an AtomPub representation for the payload of an OData request/response. |
| [[OData:JSON]](http://www.odata.org/developers/protocols/json-format) | Defines a JSON representation for the payload of an OData request/response. |
| [[OData:Batch]](http://www.odata.org/developers/protocols/batch) | Extends the OData Operations specification to define a mechanism to enable a client of a data service to "batch" a group of requests and send that group/batch to the OData service in a single HTTP request. |

OData is designed to be modular such that an OData implementation needs to implement only as much of an OData specification as required for its target scenario.

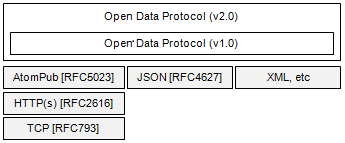
A full list of terms used by the Open Data Protocol is available on the [[OData: Terms]](http://www.odata.org/developers/protocols/terminology) page.

**Table of Contents**

1. [1](http://www.odata.org/developers/protocols/overview#RelationshipToOtherProtocols). Relationship to Other Protocols
2. [2](http://www.odata.org/developers/protocols/overview#ODataBasics). OData Basics
3. [3](http://www.odata.org/developers/protocols/overview#MetadataForODataServices). Metadata for OData Services
4. [4](http://www.odata.org/developers/protocols/overview#AbstractDataModel). Abstract Data Model
5. [4.1](http://www.odata.org/developers/protocols/overview#EntityDataModel). Entity Data Model (EDM) Overview
6. [5](http://www.odata.org/developers/protocols/overview#ServiceMetadataDocument). Service Metadata Document
7. [6](http://www.odata.org/developers/protocols/overview#AbstractTypeSystem). Primitive Type System
8. [7](http://www.odata.org/developers/protocols/overview#ProtocolVersioning). Protocol Versioning
9. [7.1](http://www.odata.org/developers/protocols/overview#CapabilityNegotiation). Capability Negotiation
10. [7.1.1](http://www.odata.org/developers/protocols/overview#RequestVersioning). Request Versioning
11. [7.1.2](http://www.odata.org/developers/protocols/overview#ResponseVersioning). Response Versioning
12. [8](http://www.odata.org/developers/protocols/overview#SecurityConsiderations). Security Considerations

**1. Relationship to Other Protocols**

As shown in Figure 1, OData builds on the conventions defined in the [Atom Publishing Protocol (AtomPub)](http://tools.ietf.org/html/rfc5023) and applies additional Web technologies such as [HTTP](http://tools.ietf.org/html/rfc2616) and [JavaScript Object Notation (JSON)](http://tools.ietf.org/html/rfc4627) to create a protocol that enables access to information from a variety of applications, services, and stores.



**2. OData Basics**

At the core of OData are **feeds**, which are **Collections** of typed **Entries**. Each entry represents a structured record with a key that has a list of **Properties** of primitive or complex types. Entries can be part of a type hierarchy and may have related entries and related feeds through **Links**. For example the following URI represents a feed of Product entries: <http://services.odata.org/OData/OData.svc/Products>.

Some entries are special in that they describe a media element (typically a BLOB), becoming two related resources: the **Media Link Entry** containing the structured data that describes the BLOB and the **Media Resource** that is the BLOB itself.

Simple OData services may consist of just a feed. More sophisticated services can have several feeds, and in that case it is useful to expose a service document that lists all the top-level feeds so clients can discover them and find out the addresses of each of them. For example, this URI <http://services.odata.org/OData/OData.svc>, identifies the service document for a sample OData service.

In addition to feeds and entries, OData services can expose **Service Operations**, which are simple, service-specific functions that accept input parameters and return entries or complex/primitive values.

As shown by the examples noted above, OData services expose all these constructs (feeds, entries, properties within entries, links, service documents, and metadata documents) via URIs in one or more representation formats, which can be acted upon (query, update, insert, delete, and so on) by clients using basic HTTP requests.

In order to help clients discover the shape of an OData service, the structure of its resources, the known links between resources, and the Service Operations exposed, an OData service may also expose a **Service Metadata Document**. OData metadata documents describe the Entity Data Model (EDM) for a given service, which is the underlying abstract data model used by OData services to formalize the description of the resources it exposes. For example, the URI <http://services.odata.org/OData/OData.svc/$metadata> identifies the metadata document for a sample OData service.

**3. Metadata for OData Services**

OData services may provide two types of metadata documents to describe themselves.

As described above in [OData Basics](http://www.odata.org/developers/protocols/overview#ODataBasics), expose a **Service Document** that lists all the top-level feeds so clients can discover them and find out the addresses of each of them. The service document is typically available at the [Service Root URI](http://www.odata.org/developers/protocols/uri-conventions#ServiceRootUri) and may be formatted in Atom or JSON as described in [[OData: Atom]](http://www.odata.org/developers/protocols/atom-format) and [[OData: JSON]](http://odataweb/%7BlocalLink:1229%7D).

All data services may also expose a [Service Metadata Document](http://www.odata.org/developers/protocols/overview#ServiceMetadataDocument) that describes the data model (i.e. structure and organization of all the resources) exposed as HTTP endpoints by the service. The following sections describe the underlying data model used by OData services and its representation - the [Service Metadata Document](http://www.odata.org/developers/protocols/overview#ServiceMetadataDocument).

**4. Abstract Data Model**

This section provides a high-level description of the EDM, which is the underlying abstract data model used by OData services.

The use of the EDM as an underlying data model for the Open Data Protocol does not mandate that a particular data persistence format or implementation be used by an OData service. The only requirement to be an OData service is that the HTTP interface exposed by the service is consistent with the protocol described in this and the [associated documents](http://www.odata.org/developers/protocols/overview#Introduction).

An OData [Service Metadata Document](http://www.odata.org/developers/protocols/overview#ServiceMetadataDocument) describes its data in EDM terms using an XML language for describing models called the conceptual schema definition language (CSDL). The remainder of this section provides a brief description of the Entity Data Model and defines how EDM constructs are mapped to the resource types (feed, entry, media link entry, service operation, and so forth) described in the [OData Basics](http://www.odata.org/developers/protocols/overview#ODataBasics) section above.

**4.1. Entity Data Model (EDM) Overview**

The central concepts in the EDM are entities and associations. **Entities** are instances of **Entity Types** (for example, Customer, Employee, and so on) which are structured records consisting of named and typed properties and with a key. **Complex Types** are structured types also consisting of a list of properties but with no key, and thus can only exist as a property of a containing entity or as a temporary value. An **Entity Key** is formed from a subset of properties of the Entity Type. The Entity Key (for example, CustomerId or OrderId) is a fundamental concept for uniquely identifying instances of Entity Types and allowing Entity Type instances to participate in relationships. Entities are grouped in **Entity Sets** (for example, Customers is a set of Customer Entity Type instances).

**Associations** define the relationship between two or more Entity Types (for example, Employee WorksFor Department). Instances of associations are grouped in **Association Sets**. **Navigation Properties** are special properties on Entity Types which are bound to a specific association and can be used to refer to associations of an entity.

Finally, all instance containers (Entity Sets and Association Sets) are grouped in an **Entity Container**.

Putting the above paragraphs into OData terms, the feeds exposed by an OData service are represented by **Entity Sets** or a Navigation Property on an Entity Type that identifies a collection of entities. For example, the Entity Set identified by the URI http://services.odata.org/OData/OData.svc/Products or the collection of entities identified by the "Products" navigation property in http://services.odata.org/OData/OData.svc/Categories(1)/Products identifies a feed of entries exposed by the OData service.

Each Entry of an OData feed is described in the EDM by an **Entity Type** and each link between entries are described by a **Navigation Property**. OData resources are described in the table below.

|  |  |
| --- | --- |
| **OData Resource** | **Is Described in an Entity Data Model by** |
| Collection | * Entity Set * A navigation property on an entity type that identifies a collection of entities |
| Entry | * Entity Type * Note: Entity Types may be part of a type hierarchy |
| Property of an entry | * Primitive or Complex Entity Type Property |
| Complex Type | * Complex Type |
| Link | * A Navigation Property defined on an Entity Type |
| Service Operation | * Function Import |

**5. Service Metadata Document**

A Service Metadata Document describes the [data model](http://www.odata.org/developers/protocols/overview#AbstractDataModel) (i.e. structure and organization of all the resources) exposed as HTTP endpoints by the service. A Service Metadata Document describes its data in [EDM](http://www.odata.org/developers/protocols/overview#EntityDataModel) terms using an XML language for describing models called the Conceptual Schema Definition Language (CSDL). CSDL is fully described in [[CSDL]](http://www.odata.org/media/6652/%5bmc-csdl%5d%5b1%5d.htm). When exposed by an OData service as a Service Metadata Document, the CSDL document is packed using the format described in [[EDMX]](http://www.odata.org/media/5986/%5bmc-edmx%5d.htm).

An example Service Metadata Document that describes three Entity Types (Categories, Products and Suppliers), the relationships among them and one "ProductsByRating" Service Operation is accessible at <http://services.odata.org/OData/OData.svc/$metadata> .

CSDL annotations are used to describe OData specific extensions to CSDL. Those annotations (represented as attributes in the [OData metadata namespace](http://www.odata.org/developers/protocols/terminology#ODataMetadataNS)) are described in the table below. Note: additional Atom format specific CSDL annotations are defined in [[OData: Atom]](http://www.odata.org/developers/protocols/atom-format).

|  |  |
| --- | --- |
| **Annotation** | **Description** |
| IsDefaultEntityContainer | A CSDL document may include many [Entity Containers](http://www.odata.org/developers/protocols/terminology#EntityContainer); this attribute is used by data services to indicate the default container. As described in [[OData:URI]](http://www.odata.org/developers/protocols/uri-conventions), Entities in the default container do not need to be container-qualified when addressed in URIs.  This attribute may be present on any element in a CSDL document  Values: true | false |
| DataServiceVersion | Indicates the version of the OData annotations used in the [Service Metadata Document](http://www.odata.org/developers/protocols/overview#ServiceMetadataDocument). Consumers of a data service metadata endpoint should first read this attribute value to determine if they can safely interpret all constructs within the document.  This attribute should be present on all elements. |
| HasStream | This attribute is used on an element to state that the Entity Type is describing a Media Link Entry in the associated OData service.  Values: true | false |
| MimeType | This attribute is used on a element to state the mime type of the property value. This mime type is used as the value of the Content-Type response header when the ["raw" value](http://www.odata.org/developers/protocols/uri-conventions#AddressingEntries) of the property is retrieved. |
| HttpMethod | This attribute is used on a element which describes a Service Operation exposed by the OData service. The value of this attribute specifies the HTTP method to be used when [invoking the Service Operation](http://www.odata.org/developers/protocols/operations#InvokeServiceOperation) as described in [[OData:Operations]](http://www.odata.org/developers/protocols/operations). |

**6. Primitive Data Types**

The Abstract Type System used to define the primitive types supported by OData is defined in detail in [MC-CSDL] (section 2.2.1). Table 2 summaries the set of primitive types supported as well as how each MUST be represented when used in an OData URI or HTTP header. Primitive type representations in request and response payloads are defined in the [[OData:Atom]](http://www.odata.org/developers/protocols/atom-format) and [[OData:JSON]](http://www.odata.org/developers/protocols/json-format) specifications.

|  |  |  |
| --- | --- | --- |
| **Primitive Types** | **Literal Form** | **Example** |
| **Null**  Represents the absence of a value | null | Example 1: null |
| **Edm.Binary**  Represent fixed- or variable- length binary data | binary'[A-Fa-f0-9][A-Fa-f0-9]\*' OR X '[A-Fa-f0-9][A-Fa-f0-9]\*' NOTE: X and binary are case sensitive. Spaces are not allowed between binary and the quoted portion. Spaces are not allowed between X and the quoted portion. Odd pairs of hex digits are not allowed. | Example 1: X'23AB' Example 2: binary'23ABFF' |
| **Edm.Boolean**  Represents the mathematical concept of binary-valued logic | true | false | Example 1: true Example 2: false |
| **Edm.Byte**  Unsigned 8-bit integer value | [A-Fa-f0-9]+ | Example 1: FF |
| **Edm.DateTime**  Represents date and time with values ranging from 12:00:00 midnight, January 1, 1753 A.D. through 11:59:59 P.M, December 9999 A.D. | datetime'yyyy-mm-ddThh:mm[:ss[.fffffff]]' NOTE: Spaces are not allowed between datetime and quoted portion. datetime is case-insensitive | Example 1: datetime'2000-12-12T12:00' |
| **Edm.Decimal**  Represents numeric values with fixed precision and scale. This type can describe a numeric value ranging from negative 10^255 + 1 to positive 10^255 -1 | decimal'[0-9]+.[0-9]+' | Example 1: decimal'2.345' |
| **Edm.Double**  Represents a floating point number with 15 digits precision that can represent values with approximate range of Â± 2.23e -308 through Â± 1.79e +308 | [0-9]+ ((.[0-9]+) | [E[+ | -][0-9]+]) | Example 1: 1E+10 Example 2: 2.029 Example 3: 2.0 |
| **Edm.Single**  Represents a floating point number with 7 digits precision that can represent values with approximate range of Â± 1.18e -38 through Â± 3.40e +38 | [0-9]+.[0-9]+f | Example 1: 2.0f |
| **Edm.Guid**  Represents a 16-byte (128-bit) unique identifier value | guid'dddddddd-dddd-dddd-dddd-dddddddddddd' where each d represents [A-Fa-f0-9] | Example 1: guid'12345678-aaaa-bbbb-cccc-ddddeeeeffff' |
| **Edm.Int16**  Represents a signed 16-bit integer value | [-][0-9]+ | Example 1: 16 Example 2: -16 |
| **Edm.Int32**  Represents a signed 32-bit integer value | [-] [0-9]+ | Example 1: 32 Example 2: -32 |
| **Edm.Int64**  Represents a signed 64-bit integer value | [-] [0-9]+L | Example 1: 64L Example 2: -64L |
| **Edm.SByte**  Represents a signed 8-bit integer value | [-] [0-9]+ | Example 1: 8 Example 2: -8 |
| **Edm.String**  Represents fixed- or variable-length character data | '<any UTF-8 character>' Note: See definition of UTF8-char in [[RFC3629]](http://tools.ietf.org/html/rfc3629) | Example 1: 'Hello OData' |
| **Edm.Time**  Represents the time of day with values ranging from 0:00:00.x to 23:59:59.y, where x and y depend upon the precision | time'<timeLiteral>' timeLiteral = Defined by the lexical representation for time at <http://www.w3.org/TR/xmlschema-2> | Example 1: 13:20:00 |
| **Edm.DateTimeOffset**  Represents date and time as an Offset in minutes from GMT, with values ranging from 12:00:00 midnight, January 1, 1753 A.D. through 11:59:59 P.M, December 9999 A.D | datetimeoffset'<dateTimeOffsetLiteral>' dateTimeOffsetLiteral = Defined by the lexical representation for datetime (including timezone offset) at <http://www.w3.org/TR/xmlschema-2> | Example 1: 2002-10-10T17:00:00Z |

**7. Protocol Versioning**

This document defines versions 1.0 and 2.0 of the Open Data Protocol (OData). Version 2.0 is a superset of the functionality available in Version 1.0. The majority of this document applies to both versions of OData. Any constructs or semantics that only exist in v2.0 are explicitly denoted as such.

OData is comprised of URL conventions, payload conventions, and HTTP interaction semantics. The OData protocol rules governing these three areas are versioned in a single OData protocol version number. The OData versioning scheme ensures that clients and servers do not misinterpret a request/response and that servers work as seamlessly as possible with clients that support a lower version number of the protocol. OData does not require all clients and servers that wish to interact to speak the same version of the OData protocol.

**7.1. Capability Negotiation**

OData supports limited capability negotiation using the DataServiceVersion and MaxDataServiceVersion version request headers and the DataServiceVersion response header. While not mandated on each request/response, clients and servers are highly encouraged to include these headers with all requests and responses to ensure no misinterpretation occurs.

**7.1.1. Request Versioning**

On a request from the client to an OData service, the DataServiceVersion and MaxDataServiceVersion version headers may be specified. If present on the request, the DataServiceVersion header value states the version of the Open Data Protocol used by the client to generate the request. If no DataServiceVersion header is provided, then the server must assume a value equal to the maximum version number the server supports.

If present on the request, the MaxDataServiceVersion header value specifies the maximum version number the client can accept in a response. A client should set this value to the maximum version number of the protocol it is able to interpret. If the header is not present in a request, the server must assume the same version number as that specified by the DataServiceVersion header. If a DataServiceVersion header is not present, then the server should assume the client can interpret the maximum version number that the server can interpret.

When the server receives a request, it must validate that the version number specified in the DataServiceVersion header (or derived value if the header is not present) is less than or equal to the maximum version number it supports. If it is not, then the server must return a response with a 4xx response code. The server should also return a description of the error using the OData XML-based error format in or OData JSON error format, as described in [[OData:Atom]](http://www.odata.org/developers/protocols/atom-format) and [[OData:JSON]](http://www.odata.org/developers/protocols/json-format).

In addition, a server must validate that the version number specified in the MaxDataServiceVersion header (or derived value if the header is not present) is greater than or equal to the minimum version number the server needs to use to generate the response. If it is not, then the server must return an error response using the OData XML error format or OData JSON error format, as described in [[OData:Atom]](http://www.odata.org/developers/protocols/atom-format) and [[OData:JSON]](http://www.odata.org/developers/protocols/json-format).

**7.1.2. Response Versioning**

On a response from server to client, the DataServiceVersion header should be specified. The value states the OData version the server used to generate the request and that should be used by the client to determine if it can correctly interpret the response (that is, the value is not larger than the value of the MaxDataServiceVersion header sent in the associated request). The value of the header should be the lowest version of the protocol the server can use to fulfill the request.

**8. Security Considerations**

The Open Data Protocol is based on HTTP, AtomPub, and JSON and thus is subject to the security considerations applicable to each of those technologies. OData implementers are encouraged to review:

* [HTTP [RFC 2616], Section 15](http://tools.ietf.org/html/rfc2616#section-15)
* [AtomPub [RFC5023], Section 15](http://tools.ietf.org/html/rfc5023#section-15)
* [JSON [RFC4627], Section 6](http://tools.ietf.org/html/rfc4627#section-6)

The Open Data Protocol does not define a new scheme for authentication or authorization. Instead, implementers of OData services may opt to use the authentication and authorization technologies that fit best with their target scenario.

The use of authentication mechanisms to prevent the insertion or editing of resources exposed by an OData service by unknown or unauthorized clients is recommended but not required.

**OData: URI Conventions**

**Introduction**

The Open Data Protocol (OData) enables the creation of REST-based data services, which allow resources, identified using Uniform Resource Identifiers (URIs) and defined in a [data model](http://www.odata.org/developers/protocols/overview#ODataBasics), to be published and edited by Web clients using simple HTTP messages. This specification defines a set of recommended (but not required) rules for constructing URIs to identify the data and metadata exposed by an OData server as well as a set of reserved URI query string operators, which if accepted by an OData server, MUST be implemented as required by this document.

The [[OData:Atom]](http://www.odata.org/developers/protocols/atom-format) and [[OData:JSON]](http://www.odata.org/developers/protocols/json-format) documents specify the format of the resource representations that are exchanged using OData and the [[OData:Operations]](http://www.odata.org/developers/protocols/operations) document describes the actions that can be performed on the URIs (optionally constructed following the conventions defined in this document) embedded in those representations.

It is encouraged that servers follow the URI construction conventions defined in this specification when possible as such consistency promotes an ecosystem of reusable client components and libraries.

The terms used in this document are defined in the [[OData: Terms]](http://www.odata.org/developers/protocols/terminology) document.

**Table of Contents**

1. [1](http://www.odata.org/developers/protocols/uri-conventions#UriComponents). URI Components
2. [2](http://www.odata.org/developers/protocols/uri-conventions#ServiceRootURI). Service Root URI
3. [3](http://www.odata.org/developers/protocols/uri-conventions#ResourcePath). Resource Path
4. [3.1](http://www.odata.org/developers/protocols/uri-conventions#AddressingEntries). Addressing Entries
5. [3.2](http://www.odata.org/developers/protocols/uri-conventions#AddressingLinksBetweenEntries). Addressing Links Between Entries
6. [3.3](http://www.odata.org/developers/protocols/uri-conventions#AddressingServiceOperations). Addressing Service Operations
7. [4](http://www.odata.org/developers/protocols/uri-conventions#QueryStringOptions). Query String Options
8. [4.1](http://www.odata.org/developers/protocols/uri-conventions#SystemQueryOptions). System Query Options
9. [4.2](http://www.odata.org/developers/protocols/uri-conventions#OrderBySystemQueryOption). OrderBy System Query Option ($orderby)
10. [4.3](http://www.odata.org/developers/protocols/uri-conventions#TopSystemQueryOption). Top System Query Option ($top)
11. [4.4](http://www.odata.org/developers/protocols/uri-conventions#SkipSystemQueryOption). Skip System Query Option ($skip)
12. [4.5](http://www.odata.org/developers/protocols/uri-conventions#FilterSystemQueryOption). Filter System Query Option ($filter)
13. [4.6](http://www.odata.org/developers/protocols/uri-conventions#ExpandSystemQueryOption). Expand System Query Option ($expand)
14. [4.7](http://www.odata.org/developers/protocols/uri-conventions#FormatSystemQueryOption). Format System Query Option ($format)
15. [4.8](http://www.odata.org/developers/protocols/uri-conventions#SelectSystemQueryOption). Select System Query Option ($select)
16. [4.9](http://www.odata.org/developers/protocols/uri-conventions#InlinecountSystemQueryOption). Inlinecount System Query Option ($inlinecount)
17. [5](http://www.odata.org/developers/protocols/uri-conventions#CustomQueryOptions). Custom Query Options
18. [6](http://www.odata.org/developers/protocols/uri-conventions#ServiceOperationParameters). Service Operation Parameters
19. [7](http://www.odata.org/developers/protocols/uri-conventions#UriEquivalence). URI Equivalence

**1. URI Components**

A URI used by an OData service has up to three significant parts: the [service root URI](http://www.odata.org/developers/protocols/uri-conventions#ServiceRootUri), [resource path](http://www.odata.org/developers/protocols/uri-conventions#ResourcePath) and [query string options](http://www.odata.org/developers/protocols/uri-conventions#QueryStringOptions). Additional URI constructs (such as a fragment) MAY be present in a URI used by an OData service; however, this specification applies no further meaning to such additional constructs.

[URIComponents](http://www.odata.org/images/ODataUri.png)

The following are two example URIs broken down into their component parts:

http://services.odata.org/OData/OData.svc

\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

|

service root URI

http://services.odata.org/OData/OData.svc/Category(1)/Products?$top=2&$orderby=name

\\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ \\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ \\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

| | |

service root URI resource path query options

**2. Service Root URI**

The service root URI identifies the root of an OData service. The resource identified by this URI MUST be an AtomPub Service Document (as specified in [RFC5023]) and follow the OData conventions for AtomPub Service Documents (or an alternate representation of an Atom Service Document if a different format is requested). OData: JSON Format specifies such an alternate JSON-based representation of a service document. The service document is required to be returned from the root of an OData service to provide clients with a simple mechanism to enumerate all of the collections of resources available for the data service.

|  |  |
| --- | --- |
| **Example Request URI** | **OData Service URI** |
| http://services.odata.org:8080 | http://services.odata.org:8080 |
| <http://services.odata.org/OData/OData.svc/Categories> | <http://services.odata.org/OData/OData.svc/> |

**3. Resource Path**

The resource path construction rules defined in this section are optional. OData servers are encouraged to follow the URI path construction rules (in addition to the required query string rules) as such consistency promotes a rich ecosystem of reusable client components and libraries.

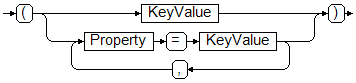
The resource path section of a URI identifies the resource to be interacted with (such as Customers, a single Customer, Orders related to Customers in London, and so forth). The resource path enables any aspect of the data model (Collections of Entries, a single Entry, Properties, Links, Service Operations, and so on) exposed by an OData service to be addressed.

**3.1. Addressing Entries**

The basic rules for addressing a Collection (of Entries), a single Entry within a Collection, as well as a property of an Entry are illustrated in the figure below.

[ResourcePath](http://www.odata.org/images/resourcePath.png)

* **Collection**: The name of a Collection or Service Operation (which returns a Collection of Entries) exposed by the service.
* **KeyPredicate**: A predicate that identifies the value(s) of the key Properties of an Entry. If the Entry has a single key Property the predicate may include only the value of the key Property. If the key is made up of two or more Properties, then its value must be stated using name/value pairs. More precisely, the syntax for a KeyPredicate is shown by the following figure.



* **NavPropSingle**: The name of a Navigation Property defined by the Entry associated with the prior path segment. The Navigation Property must identify a single entity (that is, have a "to 1" relationship).
* **NavPropCollection**: Same as NavPropSingle except it must identify a Collection of Entries (that is, have a "to many" relationship).
* **ComplexType**: The name of a declared or dynamic Property of the Entry or Complex Type associated with the prior path segment.
* **Property:** The name of a declared or dynamic Property of the Entry or Complex Type associated with the prior path segment.

For OData services conformant with the addressing conventions in this section, the canonical form of an absolute URI identifying a single Entry is formed by adding a single path segment to the service root URI. The path segment is made up of the name of the Collection associated with the Entry followed by the key predicate identifying the Entry within the Collection. For example the URIs [http://services.odata.org/OData/OData.svc/Categories(1)/Products(1)](http://services.odata.org/OData/OData.svc/Categories%281%29/Products%281%29) and [http://services.odata.org/OData/OData.svc/Products(1)](http://services.odata.org/OData/OData.svc/Products%281%29) represent the same Entry, but the canonical URI for the Entry is [http://services.odata.org/OData/OData.svc/Products(1)](http://services.odata.org/OData/OData.svc/Products%281%29).

Examples

The example URIs below follow the addressing rules stated above and are based on the reference service and its [service metadata document](http://www.odata.org/developers/protocols/overview#ServiceMetadataDocument) available at <http://services.odata.org/OData/OData.svc/> and <http://services.odata.org/OData/OData.svc/$metadata>.

<http://services.odata.org/OData/OData.svc/Categories>

* Identifies all Categories Collection.
* Is described by the Entity Set named "Categories" in the service metadata document.

[http://services.odata.org/OData/OData.svc/Categories(1)](http://services.odata.org/OData/OData.svc/Categories%281%29)

* Identifies a single Category Entry with key value 1.
* Is described by the Entity Type named "Categories" in the service metadata document.

[http://services.odata.org/OData/OData.svc/Categories(1)/Name](http://services.odata.org/OData/OData.svc/Categories%281%29/Name)

* Identifies the Name property of the Categories Entry with key value 1.
* Is described by the Property named "Name" on the "Categories" Entity Type in the service metadata document.

[http://services.odata.org/OData/OData.svc/Categories(1)/Products](http://services.odata.org/OData/OData.svc/Categories%281%29/Products)

* Identifies the collection of Products associated with Category Entry with key value 1.
* Is described by the Navigation Property named "Products" on the "Category" Entity Type in the service metadata document.

[http://services.odata.org/OData/OData.svc/Categories(1)/Products/$count](http://services.odata.org/OData/OData.svc/Categories%281%29/Products/$count)

* Identifies the number of Product Entries associated with Category 1.
* Is described by the Navigation Property named "Products" on the "Category" Entity Type in the service metadata document.

[http://services.odata.org/OData/OData.svc/Categories(1)/Products(1)/Supplier/Address/City](http://services.odata.org/OData/OData.svc/Categories%281%29/Products%281%29/Supplier/Address/City)

* Identifies the City of the Supplier for Product 1 which is associated with Category 1.
* Is described by the Property named "City" on the "Address" Complex Type in the service metadata document.

[http://services.odata.org/OData/OData.svc/Categories(1)/Products(1)/Supplier/Address/City/$value](http://services.odata.org/OData/OData.svc/Categories%281%29/Products%281%29/Supplier/Address/City/$value)

* Same as the URI above, but identifies the "raw value" of the City property.

**3.2. Addressing Links between Entries**

Much like the use of links on Web pages, the data model used by OData services supports relationships as a first class construct. For example, an OData service could expose a Collection of Products Entries each of which are related to a Category Entry.

Associations between Entries are addressable in OData just like Entries themselves are (as described above). The basic rules for addressing relationships are shown in the following figure.

http://www.odata.org/images/addressingRelationships.png

* **NavigationProperty**: The name of a Navigation Property that is declared on the Entry associated with the path segment prior to the "$links" segment.

Examples

The example URIs below follow the addressing rules stated above and are based on the reference service and its [service metadata document](http://www.odata.org/developers/protocols/overview#ServiceMetadataDocument) available at <http://services.odata.org/OData/OData.svc/> and <http://services.odata.org/OData/OData.svc/$metadata>.

[http://services.odata.org/OData/OData.svc/Categories(1)/$links/Products](http://services.odata.org/OData/OData.svc/Categories%281%29/$links/Products)

* Identifies the set of Products related to Category 1.
* Is described by the Navigation Property named "Products" on the "Category" Entity Type in the associated service metadata document.

[http://services.odata.org/OData/OData.svc/Products(1)/$links/Category](http://services.odata.org/OData/OData.svc/Products%281%29/$links/Category)

* Identifies the Category related to Product 1.
* Is described by the Navigation Property named "Category" on the "Product" Entity Type in the associated service metadata document.

**3.3. Addressing Service Operations**

OData services can expose Service Operations which, like Entries, are identified using a URI. Service Operations are simple functions exposed by an OData service whose semantics are defined by the author of the function. A Service Operation can accept primitive type input parameters and can be defined to return a single primitive, single complex type, collection of primitives, collection of complex types, a single Entry, a Collection of Entries, or void. The basic rules for constructing URIs to address Service Operations and to pass parameters to them are illustrated in the following figure.

http://www.odata.org/images/addressingServiceOperations.png

* **ServiceRootUri**: The [service root URI](http://www.odata.org/developers/protocols/uri-conventions#ServiceRootUri) identifies the root of the OData service.
* **ServiceOperation**: The name of a Service Operation exposed by an OData service.
* **ParamName**: The name of a parameter accepted by the Service Operation. If the Service Operation accepts multiple parameters, the order of the parameters in the query string of the URI is insignificant.
* **ParamValue**: The value of the parameter. The format of the value is defined by the literal form column of the table in the [Abstract Types](http://www.odata.org/developers/protocols/overview#AbstractTypeSystem) section of [[OData:Core]](http://www.odata.org/developers/protocols/overview) (the OData overview specification) .

Examples

The example URIs below follow the addressing rules stated above and are based on the reference service and its [service metadata document](http://www.odata.org/developers/protocols/overview#ServiceMetadataDocument) available at <http://services.odata.org/OData/OData.svc/> and <http://services.odata.org/OData/OData.svc/$metadata>.

[http://services.odata.org/OData/OData.svc/ProductsByColor?color='red'](http://services.odata.org/OData/OData.svc/ProductsByColor?color=%27red%27)

* Identifies the ProductByColor Service Operation and passes it a single string parameter. Since Service Operations are just functions, their semantics are up to the implementer of the function. In this case the Service Operation returns all the red Products.
* Is described by the Function Import named "ProductsByColor" that accepts a single string parameter named "color" in the service metadata document.

[http://services.odata.org/OData/OData.svc/ProductsByColor(3)/Category/Name?color='red](http://services.odata.org/OData/OData.svc/ProductsByColor%283%29/Category/Name?color=%27red)'

* Identifies the same function as the example above; however, since the function returns a collection of Entries (here, Products) it acts as a pseudo Collection in that additional path segments may follow identifying Entries or Links from the Entries within the pseudo Collection identified by the Service Operation. In this case, the result of the function is treated as a Collection of Entries, as described by the prior Addressing Entries section.
* Is described in the service metadata document by:
  + The Function Import named "ProductsByColor" that accepts a single string parameter named "color".
  + The "Category" Navigation Property on the "Product" Entity Type.
  + The "Name" property on the "Category" Entity Type.

[http://services.odata.org/OData/OData.svc/ProductsByColor?color='red'&param=foo](http://services.odata.org/OData/OData.svc/ProductsByColor?color=%27red%27&param=foo)

* Same as the example below, except an additional parameter (param) is specified. Since the function does not define an input parameter named param, this parameter is ignored and not considered part of the function invocation.

<http://services.odata.org/OData/OData.svc/ProductColors>

* Identifies the ProductColors Service Operation that accepts no parameters.
* Is described by the Function Import named "ProductColors" in the service metadata document. This function returns a collection of strings.

**4. Query String Options**

The Query Options section of an OData URI specifies three types of information: [System Query Options](http://www.odata.org/developers/protocols/uri-conventions#SystemQueryOptions), [Custom Query Options](http://www.odata.org/developers/protocols/uri-conventions#CustomerQueryOptions), and [Service Operation Parameters](http://www.odata.org/developers/protocols/uri-conventions#ServiceOperationParameters). All OData services must follow the query string parsing and construction rules defined in this section and its subsections.

**4.1. System Query Options**

System Query Options are query string parameters a client may specify to control the amount and order of the data that an OData service returns for the resource identified by the URI. The names of all System Query Options are prefixed with a "$" character.

An OData service may support some or all of the System Query Options defined. If a data service does not support a System Query Option, it must reject any requests which contain the unsupported option as defined by the request processing rules in [[OData:Operations]](http://www.odata.org/developers/protocols/operations).

**4.2. Orderby System Query Option ($orderby)**

A data service URI with a $orderby System Query Option specifies an expression for determining what values are used to order the collection of Entries identified by the Resource Path section of the URI. This query option is only supported when the resource path identifies a Collection of Entries.

The $orderby section of the normative OData specification outlines the full expression syntax supported by this query option. The examples below represent the most commonly supported subset of that expression syntax.

Examples

<http://services.odata.org/OData/OData.svc/Products?$orderby=Rating>

* All Product Entries returned in ascending order when sorted by the Rating Property.

[http://services.odata.org/OData/OData.svc/Products?$orderby=Rating asc](http://services.odata.org/OData/OData.svc/Products?$orderby=Rating%20asc)

* Same as the example above.

[http://services.odata.org/OData/OData.svc/Products?$orderby=Rating,Category/Name desc](http://services.odata.org/OData/OData.svc/Products?$orderby=Rating,Category/Name%20desc)

* Same as the URI above except the set of Products is subsequently sorted (in descending order) by the Name property of the related Category Entry.

**4.3. Top System Query Option ($top)**

A data service URI with a $top System Query Option identifies a subset of the Entries in the Collection of Entries identified by the Resource Path section of the URI. This subset is formed by selecting only the first N items of the set, where N is a positive integer specified by this query option. If a value less than 0 is specified, the URI should be considered malformed.

If the data service URI contains a $top query option, but does not contain a $orderby option, then the Entries in the set needs to first be fully ordered by the data service. While no ordering semantics are mandated, to ensure repeatable results, a data service must always use the same semantics to obtain a full ordering across requests.

Examples

<http://services.odata.org/OData/OData.svc/Products?$top=5>

* The first 5 Product Entries returned where the Collection of Products are sorted using a scheme determined by the OData service.

[http://services.odata.org/OData/OData.svc/Products?$top=5&$orderby=Name desc](http://services.odata.org/OData/OData.svc/Products?$top=5&$orderby=Name%20desc%20%20%0d)

* The first 5 Product Entries returned in descending order when sorted by the Name property.

**4.4. Skip System Query Option ($skip)**

A data service URI with a $skip System Query Option identifies a subset of the Entries in the Collection of Entries identified by the Resource Path section of the URI. That subset is defined by seeking N Entries into the Collection and selecting only the remaining Entries (starting with Entry N+1). N is a positive integer as specified by this query option. If a value less than 0 is specified, the URI should be considered malformed.

If the data service URI contains a $skip query option, but does not contain a $orderby option, then the Entries in the Collection must first be fully ordered by the data service. While no ordering semantics are mandated, to ensure repeatable results a data service must always use the same semantics to obtain a full ordering across requests.

Examples

[http://services.odata.org/OData/OData.svc/Categories(1)/Products?$skip=2](http://services.odata.org/OData/OData.svc/Categories%281%29/Products?$skip=2)

* The set of Product Entries (associated with the Category Entry identified by key value 1) starting with the third product.

<http://services.odata.org/OData/OData.svc/Products?$skip=2&$top=2&$orderby=Rating>

* The third and fourth Product Entry from the collection of all products when the collection is sorted by Rating (ascending).

**4.5. Filter System Query Option ($filter)**

A URI with a $filter System Query Option identifies a subset of the Entries from the Collection of Entries identified by the [Resource Path](http://www.odata.org/developers/protocols/uri-conventions#ResourcePath) section of the URI. The subset is determined by selecting only the Entries that satisfy the predicate expression specified by the query option.

The expression language that is used in $filter operators supports references to properties and literals. The literal values can be strings enclosed in single quotes, numbers and boolean values (true or false) or any of the additional literal representations shown in the [Abstract Type System](http://www.odata.org/developers/protocols/overview#AbstractDataModel) section.

Note: The $filter section of the normative OData specification provides an ABNF grammar for the expression language supported by this query option.

The operators supported in the expression language are shown in the following table.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| Logical Operators | | |
| Eq | Equal | /Suppliers?$filter=Address/City eq 'Redmond' |
| Ne | Not equal | /Suppliers?$filter=Address/City ne 'London' |
| Gt | Greater than | /Products?$filter=Price gt 20 |
| Ge | Greater than or equal | /Products?$filter=Price ge 10 |
| Lt | Less than | /Products?$filter=Price lt 20 |
| Le | Less than or equal | /Products?$filter=Price le 100 |
| And | Logical and | /Products?$filter=Price le 200 and Price gt 3.5 |
| Or | Logical or | /Products?$filter=Price le 3.5 or Price gt 200 |
| Not | Logical negation | /Products?$filter=not endswith(Description,'milk') |
| Arithmetic Operators | | |
| Add | Addition | /Products?$filter=Price add 5 gt 10 |
| Sub | Subtraction | /Products?$filter=Price sub 5 gt 10 |
| Mul | Multiplication | /Products?$filter=Price mul 2 gt 2000 |
| Div | Division | /Products?$filter=Price div 2 gt 4 |
| Mod | Modulo | /Products?$filter=Price mod 2 eq 0 |
| Grouping Operators | | |
| ( ) | Precedence grouping | /Products?$filter=(Price sub 5) gt 10 |

In addition to operators, a set of functions are also defined for use with the filter query string operator. The following table lists the available functions. Note: ISNULL or COALESCE operators are not defined. Instead, there is a null literal which can be used in comparisons.

|  |  |
| --- | --- |
| **Function** | **Example** |
| **String Functions** |  |
| **bool substringof(string po, string p1)** | [http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=substringof('Alfreds', CompanyName) eq true](http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=substringof%28%27Alfreds%27,%20CompanyName%29%20eq%20true) |
| **bool endswith(string p0, string p1)** | [http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=endswith(CompanyName, 'Futterkiste') eq true](http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=endswith%28CompanyName,%20%27Futterkiste%27%29%20eq%20true) |
| **bool startswith(string p0, string p1)** | [http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=startswith(CompanyName, 'Alfr') eq true](http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=startswith%28CompanyName,%20%27Alfr%27%29%20eq%20true) |
| **int length(string p0)** | [http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=length(CompanyName) eq 19](http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=length%28CompanyName%29%20eq%2019) |
| **int indexof(string p0, string p1)** | [http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=indexof(CompanyName, 'lfreds') eq 1](http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=indexof%28CompanyName,%20%27lfreds%27%29%20eq%201) |
| **string replace(string p0, string find, string replace)** | [http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=replace(CompanyName, ' ', '') eq 'AlfredsFutterkiste'](http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=replace%28CompanyName,%20%27%20%27,%20%27%27%29%20eq%20%27AlfredsFutterkiste%27) |
| **string substring(string p0, int pos)** | [http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=substring(CompanyName, 1) eq 'lfreds Futterkiste'](http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=substring%28CompanyName,%201%29%20eq%20%27lfreds%20Futterkiste%27) |
| **string substring(string p0, int pos, int length)** | [http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=substring(CompanyName, 1, 2) eq 'lf'](http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=substring%28CompanyName,%201,%202%29%20eq%20%27lf%27) |
| **string tolower(string p0)** | [http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=tolower(CompanyName) eq 'alfreds futterkiste'](http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=tolower%28CompanyName%29%20eq%20%27alfreds%20futterkiste%27) |
| **string toupper(string p0)** | [http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=toupper(CompanyName) eq 'ALFREDS FUTTERKISTE'](http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=toupper%28CompanyName%29%20eq%20%27ALFREDS%20FUTTERKISTE%27) |
| **string trim(string p0)** | [http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=trim(CompanyName) eq 'Alfreds Futterkiste'](http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=trim%28CompanyName%29%20eq%20%27Alfreds%20Futterkiste%27) |
| **string concat(string p0, string p1)** | [http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=concat(concat(City, ', '), Country) eq 'Berlin, Germany'](http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=concat%28concat%28City,%20%27,%20%27%29,%20Country%29%20eq%20%27Berlin,%20Germany%27) |
| **Date Functions** |  |
| **int day(DateTime p0)** | [http://services.odata.org/Northwind/Northwind.svc/Employees?$filter=day(BirthDate) eq 8](http://services.odata.org/Northwind/Northwind.svc/Employees?$filter=day%28BirthDate%29%20eq%208) |
| **int hour(DateTime p0)** | [http://services.odata.org/Northwind/Northwind.svc/Employees?$filter=hour(BirthDate) eq 0](http://services.odata.org/Northwind/Northwind.svc/Employees?$filter=hour%28BirthDate%29%20eq%200) |
| **int minute(DateTime p0)** | [http://services.odata.org/Northwind/Northwind.svc/Employees?$filter=minute(BirthDate) eq 0](http://services.odata.org/Northwind/Northwind.svc/Employees?$filter=minute%28BirthDate%29%20eq%200) |
| **int month(DateTime p0)** | [http://services.odata.org/Northwind/Northwind.svc/Employees?$filter=month(BirthDate) eq 12](http://services.odata.org/Northwind/Northwind.svc/Employees?$filter=month%28BirthDate%29%20eq%2012) |
| **int second(DateTime p0)** | [http://services.odata.org/Northwind/Northwind.svc/Employees?$filter=second(BirthDate) eq 0](http://services.odata.org/Northwind/Northwind.svc/Employees?$filter=second%28BirthDate%29%20eq%200) |
| **int year(DateTime p0)** | [http://services.odata.org/Northwind/Northwind.svc/Employees?$filter=year(BirthDate) eq 1948](http://services.odata.org/Northwind/Northwind.svc/Employees?$filter=year%28BirthDate%29%20eq%201948) |
| **Math Functions** |  |
| **double round(double p0)** | [http://services.odata.org/Northwind/Northwind.svc/Orders?$filter=round(Freight) eq 32](http://services.odata.org/Northwind/Northwind.svc/Orders?$filter=round%28Freight%29%20eq%2032) |
| **decimal round(decimal p0)** | [http://services.odata.org/Northwind/Northwind.svc/Orders?$filter=round(Freight) eq 32](http://services.odata.org/Northwind/Northwind.svc/Orders?$filter=round%28Freight%29%20eq%2032) |
| **double floor(double p0)** | [http://services.odata.org/Northwind/Northwind.svc/Orders?$filter=filter=round(Freight) eq 32](http://services.odata.org/Northwind/Northwind.svc/Orders?$filter=round%28Freight%29%20eq%2032) |
| **decimal floor(decimal p0)** | [http://services.odata.org/Northwind/Northwind.svc/Orders?$filter=floor(Freight) eq 32](http://services.odata.org/Northwind/Northwind.svc/Orders?$filter=floor%28Freight%29%20eq%2032) |
| **double ceiling(double p0)** | [http://services.odata.org/Northwind/Northwind.svc/Orders?$filter=ceiling(Freight) eq 33](http://services.odata.org/Northwind/Northwind.svc/Orders?$filter=ceiling%28Freight%29%20eq%2033) |
| **decimal ceiling(decimal p0)** | [http://services.odata.org/Northwind/Northwind.svc/Orders?$filter=floor(Freight) eq 33](http://services.odata.org/Northwind/Northwind.svc/Orders?$filter=floor%28Freight%29%20eq%2033) |
| **Type Functions** |  |
| **bool IsOf(type p0)** | [http://services.odata.org/Northwind/Northwind.svc/Orders?$filter=isof('NorthwindModel.Order')](http://services.odata.org/Northwind/Northwind.svc/Orders?$filter=isof%28%27NorthwindModel.Order%27%29) |
| **bool IsOf(expression p0, type p1)** | [http://services.odata.org/Northwind/Northwind.svc/Orders?$filter=isof(ShipCountry, 'Edm.String')](http://services.odata.org/Northwind/Northwind.svc/Orders?$filter=isof%28ShipCountry,%20%27Edm.String%27%29) |

**4.6. Expand System Query Option ($expand)**

A URI with a $expand System Query Option indicates that Entries associated with the Entry or Collection of Entries identified by the Resource Path section of the URI must be represented inline (i.e. eagerly loaded). For example, if you want to identify a category and its products, you could use two URIs (and execute two requests), one for /Categories(1) and one for /Categories(1)/Products. The '$expand' option allows you to identify related Entries with a single URI such that a graph of Entries could be retrieved with a single HTTP request.

The syntax of a $expand query option is a comma-separated list of Navigation Properties. Additionally each Navigation Property can be followed by a forward slash and another Navigation Property to enable identifying a multi-level relationship.

Note: The $filter section of the normative OData specification provides an ABNF grammar for the expression language supported by this query option.

Examples

<http://services.odata.org/OData/OData.svc/Categories?$expand=Products>

* Identifies the Collection of Categories as well as each of the Products associated with each Category.
* Is described by the Entity Set named "Categories" and the "Products" Navigation Property on the "Category" Entity Type in the service metadata document.

<http://services.odata.org/OData/OData.svc/Categories?$expand=Products/Suppliers>

* Identifies the Collection of Categories as well as each of the Products associated with each Category. In addition, the URI also indentifies the Suppliers associated with each Product.
* Is described by the Entity Set named "Categories", the "Products" Navigation Property on the "Category" Entity Type, and the "Suppliers" Navigation Property on the "Product" Entity Type in the service metadata document.

<http://services.odata.org/OData/OData.svc/Products?$expand=Category,Suppliers>

* Identifies the set of Products as well as the category and suppliers associated with each product.
* Is described by the Entity Set named "Products" as well as the "Category" and "Suppliers" Navigation Property on the "Product" Entity Type in the service metadata document.

**4.7. Format System Query Option ($format)**

A URI with a $format System Query Option specifies that a response to the request MUST use the media type specified by the query option. If the $format query option is present in a request URI it takes precedence over the value(s) specified in the Accept request header. Valid values for the $format query string option are listed in the following table.

|  |  |
| --- | --- |
| **$format Value** | **Response Media Type** |
| Atom | application/atom+xml |
| Xml | application/xml |
| Json | application/json |
| Any other IANA-defined content type | Any IANA-defined content type |
| A service-specific value indicating a format specific to the specific OData service | Any IANA-defined content type |

Examples

<http://services.odata.org/OData/OData.svc/Products?$format=atom>

* Identifies all Product Entries represented using the AtomPub format as defined in [[OData:Atom]](http://www.odata.org/developers/protocols/atom-format)

<http://services.odata.org/OData/OData.svc/Products?$format=json>

* Identifies all Product Entries represented using the JSON format as defined in [[OData:JSON]](http://www.odata.org/developers/protocols/json-format)

**4.8. Select System Query Option ($select)**

A data service URI with a $select System Query Option identifies the same set of Entries as a URI without a $select query option; however, the value of $select specifies that a response from an OData service should return a subset of the Properties which would have been returned had the URI not included a $select query option.

Version Note: This query option is only supported in OData version 2.0 and above.

The value of a $select System Query Option is a comma-separated list of selection clauses. Each selection clause may be a Property name, Navigation Property name, or the "\*" character. The following set of examples uses the data sample data model available at <http://services.odata.org/OData/OData.svc/$metadata> to describe the semantics for a base set of URIs using the $select system query option. From these base cases, the semantics of longer URIs are defined by composing the rules below.

Examples

<http://services.odata.org/OData/OData.svc/Products?$select=Price,Name>

* In a response from an OData service, only the Price and Name Property values are returned for each Product Entry within the Collection of products identified.
* If the $select query option had listed a Property that identified a Complex Type, then all Properties defined on the Complex Type must be returned.

<http://services.odata.org/OData/OData.svc/Products?$select=Name,Category>

* In a response from an OData service only the Name Property value and a link to the related Category Entry should be returned for each product.

<http://services.odata.org/OData/OData.svc/Categories?$select=Name,Products&$expand=Products/Suppliers>

* In a response from an OData service, only the Name of the Category Entries should be returned, but all the properties of the Entries identified by the Products and Suppliers Navigation Properties should be returned.

[http://services.odata.org/OData/OData.svc/Products?$select=\*](http://services.odata.org/OData/OData.svc/Products?$select=*)

* In a response from an OData service, all Properties are returned for each Product Entry within the Products Entity Set.
* Note: The star syntax is used to reference all properties of the Entry or Collection of Entries identified by the path of the URI or all properties of a Navigation Property. In other words, the "\*" syntax causes all Properties on an Entry to be included without traversing associations.

<http://services.odata.org/OData/OData.svc/Categories?$select=Name,Products&$expand=Products/Suppliers>

* In a response from an OData service, the Name property is included and Product Entries with all Properties are included; however, rather than including the fully expanded Supplier Entries referenced in the expand clause, each Product will contain a link that references the corresponding Collection of Supplier Entries.

Note: The $select section of the normative OData specification provides an ABNF grammar for the expression language supported by this query option.

**4.9. Inlinecount System Query Option ($inlinecount)**

A URI with a $inlinecount System Query Option specifies that the response to the request includes a count of the number of Entries in the Collection of Entries identified by the [Resource Path](http://www.odata.org/developers/protocols/uri-conventions#ResourcePath) section of the URI. The count must be calculated after applying any [$filter System Query Options](http://www.odata.org/developers/protocols/uri-conventions#FilterSystemQueryOption) present in the URI. The set of valid values for the $inlinecount query option are shown in the table below. If a value other than one shown in Table 4 is specified the URI is considered malformed.

Version Note: This query option is only supported in OData version 2.0 and above

|  |  |
| --- | --- |
| **$inlinecount value** | **Description** |
| **allpages** | The OData MUST include a count of the number of entities in the collection identified by the URI (after applying any $filter System Query Options present on the URI) |
| **none** | The OData service MUST NOT include a count in the response. This is equivalence to a URI that does not include a $inlinecount query string parameter. |

Examples

<http://services.odata.org/OData/OData.svc/Products?$inlinecount=allpages>

* Identifies all Product Entries and the count of all products.

[http://services.odata.org/OData/OData.svc/Products?$inlinecount=allpages&$top=10&Price gt 200](http://services.odata.org/OData/OData.svc/Products?$inlinecount=allpages&$top=10&Price%20gt%20200)

* Identifies the first 10 Product Entries that cost more than 200 and includes a count of the total number of Product Entries that cost more than 200.

**5. Custom Query Options**

Custom Query Options provide an extension point for OData service-specific information to be placed in the query string portion of a URI. A Custom Query String option is defined as any name/value pair query string parameter where the name of the parameter does not begin with the "$" character. Any URI exposed by an OData service may include one or more Custom Query Options.

Examples

<http://services.odata.org/OData/OData.svc/Products?x=y>

* Identifies all Product entities. Includes a Custom Query Option "x" whose meaning is service specific.

**6. Service Operation Parameters**

Service Operations represent functions exposed by an OData service. These functions may accept zero or more primitive type parameters. If a Service Operation requires an input parameter those parameters are passed via query string name/value pairs appended to the URI which identify the Service Operation as described in the [Addressing Service Operations](http://www.odata.org/developers/protocols/uri-conventions#AddressingServiceOperations) section. For nullable type parameters, a null value may be specified by not including the parameter in the query string of the URI.

Examples

<http://services.odata.org/OData/OData.svc/GetProductsByRating?rating=5>

* Identifies the "GetProductsByRating" Service Operation and specifies a value of 5 for the "rating" input parameter.

**7. URI Equivalence**

When determining if two URIs are equivalent, each URI SHOULD be normalized using the rules specified in [[RFC3987]](http://tools.ietf.org/html/rfc3987) and [[RFC3986]](http://tools.ietf.org/html/rfc3986) and then compared for equality using the equivalence rules specified in [HTTP [RFC 2616], Section 3.2.3](http://tools.ietf.org/html/rfc2616#section-3.2.3).

**OData: Operations**

**Introduction**

The OData protocol exposes a uniform service interface to operate on collections of structured and unstructured data. Most of the semantics of operations in OData come from the AtomPub protocol [RFC5023], which in turn builds on top of HTTP [[RFC2616]](http://www.ietf.org/rfc/rfc2616.txt).

This document describes the operation model for the protocol, specifying the interactions between clients and servers for retrieving and manipulating data in an OData service. It builds on the [[OData-Core]](http://www.odata.org/developers/protocols/overview) document for core concepts, [[OData-URI]](http://www.odata.org/developers/protocols/uri-conventions) for URI conventions and on the formats specifications for Atom [[OData-Atom]](http://www.odata.org/developers/protocols/atom-format) and JSON [[OData-JSON]](http://www.odata.org/developers/protocols/json-format) for the description of data representations.

**Table of Contents**

* [1](http://www.odata.org/developers/protocols/operations#Background). Background
* [1.1](http://www.odata.org/developers/protocols/operations#RepresentationFormatsAndContentTypeNegotiation) Representation formats and content type negotiation
* [1.2](http://www.odata.org/developers/protocols/operations#AdditionalconsiderationsontheuseofHTTP) Additional considerations on the use of HTTP
* [1.3](http://www.odata.org/developers/protocols/operations#Error%20Conditions) Error Conditions
* [1.4](http://www.odata.org/developers/protocols/operations#ProtocolVersioning) Protocol Versioning
* [2](http://www.odata.org/developers/protocols/operations#Operations). Operations
* [2.1](http://www.odata.org/developers/protocols/operations#RetrievingfeedsEntriesandservicedocument) Retrieving feeds, Entries and service document
* [2.2](http://www.odata.org/developers/protocols/operations#Retrievingindividualproperties) Retrieving individual properties
* [2.3](http://www.odata.org/developers/protocols/operations#Retrievingthemetadatadocument) Retrieving the metadata document
* [2.4](http://www.odata.org/developers/protocols/operations#CreatingnewEntries) Creating new Entries
* [2.5](http://www.odata.org/developers/protocols/operations#CreatingMediaLinkEntries) Creating Media Link Entries (MLEs)
* [2.6](http://www.odata.org/developers/protocols/operations#UpdatingEntries) Updating Entries
* [2.7](http://www.odata.org/developers/protocols/operations#Updatingindividualproperties) Updating individual properties
* [2.8](http://www.odata.org/developers/protocols/operations#DeletingEntries) Deleting Entries
* [2.9](http://www.odata.org/developers/protocols/operations#ManipulatingLinks) Manipulating Links
* [2.10](http://www.odata.org/developers/protocols/operations#CreatingLinksbetweenEntries) Creating Links between Entries
* [2.11](http://www.odata.org/developers/protocols/operations#RemovingLinksbetweenEntries) Removing Links between Entries
* [2.12](http://www.odata.org/developers/protocols/operations#ReplacingLinksbetweenEntries) Replacing Links between Entries
* [2.13](http://www.odata.org/developers/protocols/operations#InvokingServiceOperations) Invoking Service Operations
* [3](http://www.odata.org/developers/protocols/operations#AdditionalInteractionModelConsiderations). Additional Interaction Model Considerations
* [3.1](http://www.odata.org/developers/protocols/operations#ConcurrencycontrolandETags) Concurrency control and ETags
* [3.2](http://www.odata.org/developers/protocols/operations#RemovingLinksbetweenEntries) Removing Links between Entries
* [3.3](http://www.odata.org/developers/protocols/operations#Processingentriesofopentypes) Processing entries of open types

**1. Background**

**1.1 Representation formats and content type negotiation**

OData supports two formats for representing resources, the XML-based Atom format and the JSON format. As described in the HTTP specification [[RFC2616]](http://www.ietf.org/rfc/rfc2616.txt), clients can indicate their preference of resource representation by including an *accept* request header with a list of MIME types it can handle.

A client that wants only JSON responses would set this header to "application/json". For example:

GET /OData/OData.svc/Products HTTP/1.1

host: services.odata.org

accept: application/json

For the Atom format there is more than one MIME type involved. Atom feeds and Entries use a content type of "application/atom+xml". However, when addressing Links or properties within an element the returned resource is just XML without the Atom Entry wrapper, and its content type is "application/xml". AtomPub also introduces service documents, with a content-type of "application/atomsvc+xml". Clients can choose to support only specific content-types or all of them. To indicate it can handle all of them a client would set the *accept* header to "application/atom+xml, application/atomsvc+xml, application/xml". For example:

GET /OData/OData.svc/Products HTTP/1.1

host: services.odata.org

accept: application/atom+xml,application/atomsvc+xml,application/xml

Servers are expected to honor the client request for a format or fail with status code 406 (Not Acceptable). While HTTP 1.1 allows servers to return alternate representations when none of the requested ones are available, OData servers should return one of the requested representations or fail.

If a request does not have an *accept* header servers should default to the Atom representation.

Since in certain scenarios it is not possible for clients to control request headers, OData also has an optional query string option called $format that can be used to override the value of the request header (e.g. $format=json or $format=atom). This option is further described in [[OData-URI]](http://www.odata.org/developers/protocols/uri-conventions).

When sending a resource as part of a request or a response body, clients and servers should set the corresponding MIME type in the *content-type* header.

**1.2 Additional considerations on the use of HTTP**

OData heavily builds on HTTP for its interaction model. In addition to content-type negotiation clients and servers should use HTTP mechanisms for aspects such as character set negotiation (through accept-charset and content-type headers), caching and redirection.

Similarly, while this specification uses specific status codes for specific outcomes for a request, clients should be prepared to handle all HTTP status codes and interpret them appropriately.

**1.3 Error Conditions**

Error conditions are in general exposed as HTTP responses with error status codes (4xx and 5xx). Individual descriptions of operations in this specification do not call out error conditions. Servers should map error conditions to HTTP status codes. Common examples include using 404 (Not Found) when receiving requests against a URI not defined by this server, 400 for a general error in the request and 500 for a server-side error while processing the request.

Clients should be prepared to handle HTTP error codes in all requests.

In addition to the status code, servers should include a response body for error responses that includes more information about the error condition. The level of detail and nature of the information should be appropriate for the level of exposure of the service so as not to expose sensitive information to untrusted clients. The format for error messages is described in [[OData-Atom]](http://www.odata.org/developers/protocols/atom-format) and [[OData-JSON]](http://www.odata.org/developers/protocols/json-format).

**1.4 Protocol Versioning**

In order to provide a safe mechanism for evolution of the protocol OData has a versioning scheme. While OData is designed such that it can operate to certain extent without the use of version headers, clients and servers are strongly encouraged to follow the protocol versioning scheme. Details of OData versioning are covered in [[OData-Core]](http://www.odata.org/developers/protocols/overview).

All the examples in this document assume clients and servers that use version 2.0 of the protocol.

**2. Operations**

The OData service interface has a fixed number of operations that have uniform meaning across all the resources it can act on. These operations are retrieve, create, update and delete and they map to the GET, POST, PUT/MERGE and DELETE HTTP methods. Each of them acts on a resource that is indicated using a URI. In addition to the uniform interface operations, OData allows servers to expose custom operations (known as Service Operations) that can be invoked through GET or POST.

**2.1 Retrieving feeds, Entries and service document**

Clients retrieve a feed, Entry or service document by issuing an HTTP GET request against its URI. Servers respond with the feed, Entry or service document in the response body in the proper format. For example, to request the feed of products in the Atom-based format from an example product catalog OData service and its corresponding response:

Request:

GET /OData/OData.svc/Products HTTP/1.1

Host: services.odata.org

accept: application/atom+xml,application/xml

DataServiceVersion: 1.0

MaxDataServiceVersion: 2.0

Response:

HTTP/1.1 200 OK

Content-Length: 5685

Date: Sat, 27 Feb 2010 20:03:28 GMT

Content-Type: application/atom+xml;charset=utf-8

DataServiceVersion: 1.0;

<?xml version="1.0" encoding="utf-8" standalone="yes"?>

<feed xml:base="http://services.odata.org/OData/OData.svc/"

xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"

xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"

xmlns="http://www.w3.org/2005/Atom">

<title type="text">Products</title>

<id>http://services.odata.org/OData/OData.svc/Products</id>

<updated>2010-02-27T20:03:28Z</updated>

<Link rel="self" title="Products" href="Products" />

<Entry>

...

</Entry>

<Entry>

...

</Entry>

<Entry>

...

</Entry>

</feed>

To obtain a JSON response a client would issue the same request with a different *accept* header:

Request:

GET /OData/OData.svc/Products HTTP/1.1

Host: services.odata.org

accept: application/json

DataServiceVersion: 1.0

MaxDataServiceVersion: 2.0

Response:

HTTP/1.1 200 OK

Content-Length: 2481

Date: Sat, 27 Feb 2010 20:08:33 GMT

Content-Type: application/json;charset=utf-8

DataServiceVersion: 1.0;

{

"d" : [

{ ... },

{ ... },

{ ... },

]

}

When a Collection is too large to be returned in a single response, servers can return a *partial list* of entries as defined in AtomPub. A feed for a partial list contains subset of the entries of the Collection and a link to the next (potentially partial) list. For example:

Request:

GET /Northwind/Northwind.svc/Customers HTTP/1.1

Host: services.odata.org

accept: application/atom+xml,application/xml

DataServiceVersion: 1.0

MaxDataServiceVersion: 2.0

Response:

HTTP/1.1 200 OK

Content-Length: 28735

Date: Fri, 12 Mar 2010 18:05:49 GMT

Content-Type: application/atom+xml;charset=utf-8

DataServiceVersion: 2.0;

<?xml version="1.0" encoding="utf-8" standalone="yes"?>

<feed xml:base="http://services.odata.org/Northwind/Northwind.svc/"

xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"

xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"

xmlns="http://www.w3.org/2005/Atom">

<title type="text">Customers</title>

<id>http://services.odata.org/Northwind/Northwind.svc/Customers</id>

<updated>2010-03-12T18:05:49Z</updated>

<link rel="self" title="Customers" href="Customers" />

<entry>

...

</entry>

<entry>

...

</entry>

<entry>

...

</entry>

<link rel="next"

href="http://services.odata.org/Northwind/Northwind.svc/Customers?$skiptoken='ERNSH'" />

</feed>

There are no expectations about stability of the data across requests while traversing partial lists. Clients should we prepared to deal with missing or repeating entries.

While in many cases clients will directly follow links provided externally or found in a service document to find and retrieve a feed for a Collection, often a client will want to further control how the feed is returned. OData provide a series of optional conventions to allow clients to filter, sort and page over data in Collections, we well as to request for a subset of the properties to be sent and to expand related entries inline. All these options are discussed in [[OData-URI]](http://www.odata.org/developers/protocols/uri-conventions). An example on this would be (before escaping the URL):

http://services.odata.org/OData/OData.svc/Products?$filter=Rating gt 2&$orderby=Price&$select=Rating,Price

this requests all products with a Rating greater than 2, sorted by Price in ascending order, and asks the server to only retrieve the Rating and Price properties.

**2.2 Retrieving individual properties**

Servers may support retrieval of individual properties within Entries. OData provides two ways of exposing individual properties, one that follows OData formats and another one that exposes the raw value of the property.

When using the optional OData URL conventions, the former has an address that follows the containing Entry with the property name (e.g. Products(1)/Description to obtain the value of the "Description" property), and the latter uses a $value suffix (e.g. Products(1)/Description/$value). More details in the Resource Path section of the [[OData-URI]](http://www.odata.org/developers/protocols/uri-conventions) document.

To retrieve a property value using one of the OData formats a client issues an HTTP request against the property URI, and uses content-type negotiation to indicate the expected format. Since sub-Entry elements such as properties aren't represented as whole Atom Entries, clients that use the Atom format should use "application/xml" as the content type and should be prepared to handle an XML response that is not an Atom feed or Entry. This is described in further detail in [[OData-Atom]](http://www.odata.org/developers/protocols/atom-format). An example of a request to retrieve the "Description" property in a Product resource follows.

Request:

GET /OData/OData.svc/Products(1)/Description HTTP/1.1

Host: services.odata.org

accept: application/xml

DataServiceVersion: 1.0

MaxDataServiceVersion: 2.0

Response:

HTTP/1.1 200 OK

Content-Length: 158

Date: Sat, 27 Feb 2010 21:10:15 GMT

Content-Type: application/xml;charset=utf-8

DataServiceVersion: 1.0;

<?xml version="1.0" encoding="utf-8" standalone="yes"?>

<Description xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices">

Low fat milk

</Description>

To retrieve a property value in raw form a client issues an HTTP GET request against the property's raw value URL. Typically these values have a fixed content-type and negotiation is not supported (although servers are allowed to introduce this capability). Clients not having out of band knowledge of expected content-types should use "\*/\*" in their accept header (or not include an accept header in the request). For example:

Request:

GET /OData/OData.svc/Products(1)/Description/$value HTTP/1.1

Host: services.odata.org

DataServiceVersion: 1.0

MaxDataServiceVersion: 2.0

Response:

HTTP/1.1 200 OK

Content-Length: 12

Date: Sat, 27 Feb 2010 21:16:05 GMT

Content-Type: text/plain;charset=utf-8

DataServiceVersion: 1.0;

Low fat milk

**2.3 Retrieving the metadata document**

Servers may expose a metadata document that describes the structure of the service and its resources. Conventionally this document is located at the /$metadata address relative to the service root URI of the service. OData metadata documents are discussed in [[OData-Core]](http://www.odata.org/developers/protocols/overview).

Clients can obtain the metadata document by issuing an HTTP GET request against the metadata URI. Since only an XML serialization of EDM schemas exist currently, no content-type negotiation is supported for this resource. Other HTTP aspects such as caching and character set encoding still apply.

**2.4 Creating new Entries**

Following the AtomPub protocol, new Entries are created by executing an HTTP POST request against the URI of the Collection where the Entry is to be created. The POST request includes the new Entry in its body, in one of the supported formats. For example:

POST /OData/OData.svc/Categories HTTP/1.1

Host: services.odata.org

DataServiceVersion: 1.0

MaxDataServiceVersion: 2.0

accept: application/atom+xml

content-type: application/atom+xml

Content-Length: 634

<?xml version="1.0" encoding="utf-8"?>

<Entry xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"

xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"

xmlns="http://www.w3.org/2005/Atom">

<title type="text"></title>

<updated>2010-02-27T21:36:47Z</updated>

<author>

<name />

</author>

<category term="DataServiceProviderDemo.Category"

scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />

<content type="application/xml">

<m:properties>

<d:ID>10</d:ID>

<d:Name>Clothing</d:Name>

</m:properties>

</content>

</Entry>

Note that the client indicates both the content-type of the request body and the accept header to indicate the format of the response body, if any.

The server processes the request by creating the resource, assigning default values to all the properties not indicated in the request that are optional, and sending the final state of the resource back to the client in the response, including a "location" header that contains the URL of the Entry that was created. HTTP status code 201 reflects that the Entry has been created. For example:

HTTP/1.1 201 Created

Content-Length: 1072

Date: Sat, 27 Feb 2010 21:39:54 GMT

Location: http://services.odata.org/OData/OData.svc/Categories(10)

Content-Type: application/atom+xml;charset=utf-8

DataServiceVersion: 1.0;

<?xml version="1.0" encoding="utf-8" standalone="yes"?>

<Entry xml:base="http://services.odata.org/OData/OData.svc/"

xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"

xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"

xmlns="http://www.w3.org/2005/Atom">

<id>http://services.odata.org/OData/OData.svc/Categories(10)</id>

<title type="text"></title>

<updated>2010-02-27T21:39:54Z</updated>

<author>

<name />

</author>

<Link rel="edit" title="Category" href="Categories(10)" />

<Link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Products"

type="application/atom+xml;type=feed" title="Products" href="Categories(10)/Products" />

<category term="DataServiceProviderDemo.Category"

scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />

<content type="application/xml">

<m:properties>

<d:ID m:type="Edm.Int32">10</d:ID>

<d:Name>Clothing</d:Name>

</m:properties>

</content>

</Entry>

The Entry being created may contain Links to other Entries in the service. If that is the case the server is expected to create the Entry and the appropriate Links. For example, to create a new product in the catalog that is associated with the category Entry created above a client would execute a POST request against the OData.svc/Products collection with a product Entry (similar to the Category Entry above, with the proper adjustments for type information and property names and values) containing a Link to the Category Entry using any URI that resolves to that resource:

<Link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Category"

type="application/atom+xml;type=Entry"

title="Category"

href="Categories(10)" />

or the equivalent in JSON format as described in the Representing Links section of [[OData-JSON]](http://www.odata.org/developers/protocols/json-format).

Alternatively a client can create and Link an Entry to a related Entry by leveraging the addressing scheme if the server supports addressing related items. For example, if a server implements the OData URI conventions described in [[OData-URI]](http://www.odata.org/developers/protocols/uri-conventions), the address …/Categories(10)/Products points at all the products in the specified category. When a POST request is issued against that products collection (instead of the top-level products collection) the server will create the new product Entry and automatically Link it to the parent category. Clients may combine this method and the previous one to create an Entry that is related to another one implicitly through the relationship implied in the URL, and related to other Entries by explicitly-specified Links in the request body.

When a client needs to create multiple related Entries it can do so as independent operations or -if the Links between Entries allow it structurally- they can perform a single POST with a tree of Entries. The tree is formed by using inline expansion as described in [[OData-Atom]](http://www.odata.org/developers/protocols/atom-format) and [[OData-JSON]](http://www.odata.org/developers/protocols/json-format). All expanded Entries are considered new. Servers process a request with inline Entries by creating individual Entries and then Linking them in the same way Linking would have happened in an independent request. This example creates the same product category Entry as before, but also creates two products in it. While this example shows an inline feed with multiple Entries, single Entries without a wrapping feed are also allowed when the cardinality of the relationship represented by the Link allows it.

Request:

POST /OData/OData.svc/Categories HTTP/1.1

Host: services.odata.org

DataServiceVersion: 1.0

MaxDataServiceVersion: 2.0

accept: application/atom+xml

content-type: application/atom+xml

Content-Length: 2626

<?xml version="1.0" encoding="utf-8"?>

<Entry xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"

xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"

xmlns="http://www.w3.org/2005/Atom">

<title type="text"></title>

<updated>2010-02-27T21:36:47Z</updated>

<author>

<name />

</author>

<category term="DataServiceProviderDemo.Category"

scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />

<Link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Products"

type="application/atom+xml;type=feed" title="Products" href="Categories(1)/Products">

<m:inline>

<feed>

<Entry>

<title type="text"></title>

<updated>2010-02-27T21:36:47Z</updated>

<author>

<name />

</author>

<category term="DataServiceProviderDemo.Product"

scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />

<content type="application/xml">

<m:properties>

<d:ID m:type="Edm.Int32">10</d:ID>

<d:Name>T-Shirt</d:Name>

<d:Description>Just a T-Shirt</d:Description>

<d:ReleaseDate m:type="Edm.DateTime">2010-01-01T00:00:00</d:ReleaseDate>

<d:DiscontinuedDate m:type="Edm.DateTime" m:null="true" />

<d:Rating m:type="Edm.Int32">0</d:Rating>

<d:Price m:type="Edm.Decimal">15.0</d:Price>

</m:properties>

</content>

</Entry>

<Entry>

<title type="text"></title>

<updated>2010-02-27T21:36:47Z</updated>

<author>

<name />

</author>

<category term="DataServiceProviderDemo.Product"

scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />

<content type="application/xml">

<m:properties>

<d:ID m:type="Edm.Int32">11</d:ID>

<d:Name>Shorts</d:Name>

<d:Description>Running shorts</d:Description>

<d:ReleaseDate m:type="Edm.DateTime">2010-01-01T00:00:00</d:ReleaseDate>

<d:DiscontinuedDate m:type="Edm.DateTime" m:null="true" />

<d:Rating m:type="Edm.Int32">0</d:Rating>

<d:Price m:type="Edm.Decimal">21.0</d:Price>

</m:properties>

</content>

</Entry>

</feed>

</m:inline>

</Link>

<content type="application/xml">

<m:properties>

<d:ID>10</d:ID>

<d:Name>Clothing</d:Name>

</m:properties>

</content>

</Entry>

Response:

HTTP/1.1 201 Created

Content-Length: 1072

Date: Sun, 28 Feb 2010 08:59:57 GMT

Location: http://services.odata.org/OData/OData.svc/Categories(10)

Content-Type: application/atom+xml;charset=utf-8

DataServiceVersion: 1.0;

<?xml version="1.0" encoding="utf-8" standalone="yes"?>

<Entry xml:base="http://services.odata.org/OData/OData.svc/"

xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"

xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"

xmlns="http://www.w3.org/2005/Atom">

<id>http://services.odata.org/OData/OData.svc/Categories(10)</id>

<title type="text"></title>

<updated>2010-02-28T08:59:57Z</updated>

<author>

<name />

</author>

<Link rel="edit" title="Category" href="Categories(10)" />

<Link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Products"

type="application/atom+xml;type=feed" title="Products" href="Categories(10)/Products" />

<category term="DataServiceProviderDemo.Category"

scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />

<content type="application/xml">

<m:properties>

<d:ID m:type="Edm.Int32">10</d:ID>

<d:Name>Clothing</d:Name>

</m:properties>

</content>

</Entry>

**2.5 Creating Media Link Entries (MLEs)**

Media Link Entries (MLEs) are created by issuing a POST request against the collection the MLE should be created in, with the request body containing the Media Resource (MR) and the Content-Type header indicating its media type.

As described in AtomPub, servers should create both the MR and the MLE during the execution of the POST request, and return the URI of the MLE in the Location response header.

OData servers usually initialize the structured data in the MLE by using server-generated values and information extracted from the MR. The MLE can optionally be included in the response body to update client state with server-generated information. Servers may return 201 (Created) or 200 (OK) when the MLE and MR were successfully created and a response body is returned, or 204 (No Content) if creation is successful but no response is returned.

Use of the *slug* header as defined in AtomPub, section 9.7 is supported and has no further semantics than those defined in that specification.

**2.6 Updating Entries**

To update the data in an Entry clients execute an HTTP PUT request against the Entry's URI, with a new Entry resource included in the request body.

According to the AtomPub protocol specification, the PUT request replaces the existing Entry, so all property values in the Entry either take the values indicated in the request body, or are reset to their default value if not mentioned in the request. For example, to update a product in the sample OData service:

Request:

PUT /OData/OData.svc/Products(1) HTTP/1.1

Host: services.odata.org

DataServiceVersion: 1.0

MaxDataServiceVersion: 2.0

accept: application/atom+xml

content-type: application/atom+xml

Content-Length: 1181

<?xml version="1.0" encoding="utf-8" standalone="yes"?>

<Entry xml:base="http://services.odata.org/OData/OData.svc/" xmlns:d="

http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="

http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="

http://www.w3.org/2005/Atom">

<id>http://services.odata.org/OData/OData.svc/Products(1)</id>

<title type="text"></title>

<updated>2010-02-28T10:23:02Z</updated>

<author>

<name />

</author>

<Link rel="edit" title="Product" href="Products(1)" />

<category term="DataServiceProviderDemo.Product" scheme="

http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />

<content type="application/xml">

<m:properties>

<d:ID m:type="Edm.Int32">1</d:ID>

<d:Name>Milk</d:Name>

<d:Description>Low fat milk</d:Description>

<d:ReleaseDate m:type="Edm.DateTime">1995-10-21T00:00:00</d:ReleaseDate>

<d:DiscontinuedDate m:type="Edm.DateTime" m:null="true" />

<d:Rating m:type="Edm.Int32">4</d:Rating>

<d:Price m:type="Edm.Decimal">4.5</d:Price>

</m:properties>

</content>

</Entry>

Response:

HTTP/1.1 204 No Content

Date: Sun, 28 Feb 2010 11:32:26 GMT

DataServiceVersion: 1.0;

When processing a PUT request servers return status 204 (No Content) to indicate success, no response body is needed.

In certain cases it is desirable to perform an incremental update without replacing all the contents of an Entry. In order to avoid overloading the meaning of PUT, OData uses the custom HTTP method MERGE for this scenario. A MERGE request updates only the properties indicated in the request body, and leaves untouched anything not mentioned in its current state. For example, to only update the price and rating of a product:

MERGE /OData/OData.svc/Products(1) HTTP/1.1

Host: services.odata.org

DataServiceVersion: 1.0

MaxDataServiceVersion: 2.0

accept: application/atom+xml

content-type: application/atom+xml

Content-Length: 1181

<?xml version="1.0" encoding="utf-8" standalone="yes"?>

<Entry xml:base="http://services.odata.org/OData/OData.svc/" xmlns:d="

http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="

http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="

http://www.w3.org/2005/Atom">

<id>http://services.odata.org/OData/OData.svc/Products(1)</id>

<title type="text"></title>

<updated>2010-02-28T10:23:02Z</updated>

<author>

<name />

</author>

<Link rel="edit" title="Product" href="Products(1)" />

<category term="DataServiceProviderDemo.Product" scheme="

http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />

<content type="application/xml">

<m:properties>

<d:Rating m:type="Edm.Int32">4</d:Rating>

<d:Price m:type="Edm.Decimal">4.5</d:Price>

</m:properties>

</content>

</Entry>

While there is a distinction between PUT and MERGE for properties, the Links of an Entry are not directly considered part of the structured data portion of an Entry and thus are not reset on PUT. Servers should use MERGE semantics for Links for both PUT and MERGE requests.

NOTE: there is active discussion in the community about the introduction of a PATCH method in HTTP. Once PATCH is an official part of the HTTP specification OData will support PATCH and phase out the custom MERGE method over time.

**2.7 Updating individual properties**

It is also possible to update individual properties using the same addresses a client can use to retrieve them. Clients may use the PUT method to update the value of a property. Similarly for the retrieve case, clients can choose to use URIs that point to the property in its OData format or in raw form.

For example, to update only the rating of a product:

PUT /OData/OData.svc/Products(1)/Rating HTTP/1.1

Host: services.odata.org

DataServiceVersion: 1.0

MaxDataServiceVersion: 2.0

accept: application/xml

content-type: application/xml

Content-Length: 275

<?xml version="1.0" encoding="utf-8" standalone="yes"?>

<d:Rating xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"

xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"

m:type="Edm.Int32">5</d:Rating>

Alternatively clients can use the address of the raw value of a property and send data in its native form. The example below shows updating the same "Rating" property, although more natural scenarios would include updating an image or an HTML document with a newer version without the need to encode it:

PUT /OData/OData.svc/Products(1)/Rating/$value HTTP/1.1

Host: services.odata.org

DataServiceVersion: 1.0

MaxDataServiceVersion: 2.0

accept: application/xml

content-type: text/plain

Content-Length: 1

5

In either case servers should update the property and return 204 (No Content) and no response body or 200 (OK) and an update of the property as appropriate.

Since properties are considered atomic elements (no sub-property operations exist) the MERGE method is not defined for individual properties and servers should fail requests with this method with status code 405 (Method Not Allowed).

**2.8 Deleting Entries**

Entries are deleted by executing an HTTP DELETE request against a URI that points at the Entry. If the operation executed successfully servers should return 200 (OK) with no response body.

If the Entry has dependent Entries such as Entries Linked to it, it is up to the server whether deletion should be cascaded, the operation should fail because of the presence of dependent Entries, or if Links are left dangling after the Entry is deleted. This largely depends on the server, storage model and the actual application the OData service is a part of.

In the case of Media Link Entries, deleting the Media Link Entry also deletes the Media Resource.

**2.9 Manipulating Links**

OData provides two ways to manipulate Links: through Link constructs in Entries and by operating on the Link resources directly.

While manipulating Links by updating the Entries at either end can be simpler, this mechanism is less expressive and it is sometimes required to use Link addresses directly. For example, to add or remove a Link between two existing Entries where the Link has a cardinality of "many" on both ends (modeling a many-to-many relationship) requires the use of operations on Links directly.

**2.10 Creating Links between Entries**

A new Link between two Entries can be established by referencing one Entry during creation or modification of another Entry (as described in [Creating new Entries](http://www.odata.org/developers/protocols/operations#CreatingnewEntries) and [Updating Entries](http://www.odata.org/developers/protocols/operations#UpdatingEntries) ) or by explicitly issuing a POST request against the URL of the Link resource. The request must have the new URI in the request body following the appropriate format for stand-alone Links in XML or JSON as described in [[OData-Atom]](http://www.odata.org/developers/protocols/atom-format) and [[OData-JSON]](http://www.odata.org/developers/protocols/json-format).

Servers acknowledge the creation of the Link by returning a status code of 204 (No Content) and no response body. For example, to add a Link to an existing product to the list of products (through the "products" Link) of a given product category:

Request:

POST /OData/OData.svc/Categories(1)/$links/Products HTTP/1.1

Host: services.odata.org

DataServiceVersion: 1.0

MaxDataServiceVersion: 2.0

Content-Length: 159

Content-Type: application/xml

<uri xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices">

http://services.odata.org/OData/OData.svc/Products(10)

</uri>

Response:

HTTP/1.1 204 No Content

Date: Tue, 02 Mar 2010 08:59:32 GMT

DataServiceVersion: 1.0;

If using the OData URI conventions described in [[OData-URI]](http://www.odata.org/developers/protocols/uri-conventions), the Link resource address can be derived by using the convention; otherwise it needs to be obtained from the Entry itself.

Adding a Link using POST may result in only the new Link being added or it may have further side-effects. In particular, if the other end of the Link has a cardinality of 1, then the server may remove a previously existing Link (if any) in order to create this one and still satisfy the cardinality constrain.

**2.11 Removing Links between Entries**

Existing Links may be removed by executing a DELETE request against the Link URI. Servers respond with the success status code of 204 (No Content) and no response body. For example, to delete the Link between a product and it's containing product category:

Request:

DELETE /OData/OData.svc/Categories(1)/$links/Products(10) HTTP/1.1

Host: services.odata.org

Content-Length: 0

Response:

HTTP/1.1 204 No Content

Date: Tue, 02 Mar 2010 09:09:02 GMT

DataServiceVersion: 1.0;

Removing Links may cause further side-effects on the server. For example servers may cascade the deletion to the Entry or Entries the Link pointed to.

Besides regular failures due to request correctness or permissions, the server may fail a DELETE request against a Link if deleting the Link would put the service data set in an inconsistent state (i.e. a state that is not allowed by the service schema).

**2.12 Replacing Links between Entries**

Clients may replace a Link with a new Link by issuing an HTTP PUT request against the Link resource URI, with the new URI in the request body following the appropriate format for stand-alone Links in XML or JSON as described in [[OData-Atom]](http://www.odata.org/developers/protocols/atom-format) and [[OData-JSON]](http://www.odata.org/developers/protocols/json-format). Servers should update the Link if the service constraints allow it and return 204 (No Content). For example, to move a product from the product category with key "1" to the category with key "2":

Request:

PUT /OData/OData.svc/Products(1)/$links/Category HTTP/1.1

Host: services.odata.org

DataServiceVersion: 1.0

MaxDataServiceVersion: 2.0

accept: application/xml

content-type: application/xml

Content-Length: 158

<uri xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices">

http://services.odata.org/OData/OData.svc/Categories(2)

</uri>

Response:

HTTP/1.1 204 No Content

Date: Tue, 02 Mar 2010 19:54:52 GMT

DataServiceVersion: 1.0;

Note that when updating the end of a Link that has a cardinality of 1 it is also possible to perform the operation through the Entry itself. For example, to switch back the product of the previous example to its original category:

Request:

MERGE /OData/OData.svc/Products(1) HTTP/1.1

Host: services.odata.org

DataServiceVersion: 1.0

MaxDataServiceVersion: 2.0

content-type: application/atom+xml

Content-Length: 590

<entry xml:base="http://services.odata.org/OData/OData.svc/"

xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"

xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"

xmlns="http://www.w3.org/2005/Atom">

<Link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Category"

type="application/atom+xml;type=Entry"

title="Category" href="Categories(1)" />

<category term="DataServiceProviderDemo.Product"

scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />

</entry>

Response:

HTTP/1.1 204 No Content

Date: Tue, 02 Mar 2010 20:06:12 GMT

DataServiceVersion: 1.0;

Since this is a merge operation and no actual properties are mentioned, the server should only update the Links included in the request body.

**2.13 Invoking Service Operations**

Service Operations expose custom behaviors that do not map to the uniform interface. Service operations can be defined to take parameters, which are simple scalar values of one of the EDM primitive types as defined in [[OData-Core]](http://www.odata.org/developers/protocols/overview).

Clients may invoke a Service Operation by issuing an HTTP GET or POST request against their URI. If a particular Service Operation takes parameters they should be included as part of the query string in the request URI as described in the Addressing Service Operations section of [[OData-URI]](http://www.odata.org/developers/protocols/uri-conventions). The name of each parameter is used as the name of the query string parameter, and its value should be in OData literal form. No "$" prefix should be used in parameters ("$" is reserved for OData parameters). For example:

Request:

GET /odata/odata.svc/GetProductsByRating?rating=4 HTTP/1.1

Host: services.odata.org

DataServiceVersion: 1.0

MaxDataServiceVersion: 2.0

Response:

HTTP/1.1 200 OK

Content-Length: 1957

Date: Tue, 02 Mar 2010 20:59:21 GMT

Content-Type: application/atom+xml;charset=utf-8

DataServiceVersion: 1.0;

<?xml version="1.0" encoding="utf-8" standalone="yes"?>

<feed xml:base="http://services.odata.org/(S(dwaygaylwritgwuiyvfad5ln))/OData/OData.svc/"

xmln:d="http://schemas.microsoft.com/ado/2007/08/dataservices"

xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"

xmlns="http://www.w3.org/2005/Atom">

<title type="text">GetProductsByRating</title>

<id>http://services.odata.org/(S(dwaygaylwritgwuiyvfad5ln))/odata/odata.svc/GetProductsByRating</id>

<updated>2010-03-02T20:59:21Z</updated>

<Link rel="self" title="GetProductsByRating" href="GetProductsByRating" />

<Entry>

...

</Entry>

</feed>

Server implementations should only allow invocation of Service Operations through GET when the operations will not cause side-effects in the system.

Service authors can choose to enable query composition in Service Operations if the operation returns a Collection of Entries. When that is the case, clients may use the URI conventions defined in [[OData-URI]](http://www.odata.org/developers/protocols/uri-conventions) treating the URI to the Service Operation as the URI to the initial collection, and then applying the convention from there.

**3. Additional Interaction Model Considerations**

**3.1 Concurrency control and ETags**

OData uses HTTP ETags for optimistic concurrency control. A few special considerations apply for ETags:

* When retrieving an Entry the server returns an opaque ETag value
  + When getting several Entries in a feed, the ETag value is included as metadata in the Entry itself. See [[OData-Atom]](http://www.odata.org/developers/protocols/atom-format) and [[OData-JSON]](http://www.odata.org/developers/protocols/json-format) for actual format details.
  + When retrieving a single Entry, the ETag is returned as a response header called ETag as defined by HTTP. The Server can choose to also include it in the body as they would do for feeds for consistency.
  + During processing of POST, PUT and MERGE the server should compute a new ETag and return it in a response header, regardless of whether the response has a body with the actual Entry information.
* When issuing a PUT, MERGE or DELETE request, clients need to indicate an ETag in the *If-Match* HTTP request header.
  + If for a given client it is acceptable to overwrite any version of the Entry in the server, then the value "\*" may be used instead.
  + If a given Entry has an ETag and a client attempts to modify or delete the Entry without an If-Match header servers should fail the request with a 412 response code.

OData servers will use weak ETags often as a way of indicating that two resources may be semantically equivalent but a particular request may see a different representation of it. Clients should be prepared to handle weak and strong ETags.

**3.2 Method Tunneling through POST**

In many scenarios clients are limited to the HTTP GET and POST methods only. In order to help work-around this limitation, OData servers can support method tunneling through POST. The methods that can be executed through tunneling are MERGE, PUT and DELETE.

To issue a request with method tunneling a client sets up a request with body and headers as needed, but uses POST as the HTTP method instead of the actual required one. It then adds one more header, "X-HTTP-Method", and gives it the value MERGE, PUT or DELETE.

Servers must check if POST requests have the X-HTTP-Method header set to one of the valid values and if so execute the rest of the request as if the header value was the actual HTTP method for it.

**3.3 Processing entries of open types**

As discussed in [[OData-Core]](http://www.odata.org/developers/protocols/overview) the OData data model supports open types for Entries. An entry of an open type may have extra properties (dynamic properties) in addition to those statically declared in metadata.

In general servers should treat dynamic properties and static properties as uniformly as possible.

For data retrieval servers should include dynamic and static properties in entries and should use the same format for both, as discussed in [[OData-Atom]](http://www.odata.org/developers/protocols/atom-format) and [[OData-JSON]](http://www.odata.org/developers/protocols/json-format).

From the query semantics perspective dynamic and static properties apply equally in all contexts (projection, filter, sorting, etc.). Similarly in the addressing scheme dynamic and static properties should be handled in the same way (e.g. it is legal to address a dynamic property within an entry in servers that support the URI convention discussed in [[OData-URI]](http://www.odata.org/developers/protocols/uri-conventions)). Dynamic properties may not be present in all entries within a collection; query processors should treat missing properties as if they had a null value.

During update processing, dynamic properties should be appended to entries when they appear on a POST, PUT or MERGE request. They can also be incrementally added to existing entries using a MERGE operation with the dynamic properties to be added. A PUT request resets all the properties not mentioned in the request to their default values, and removes all the dynamic properties that may have been previously added to this entry but were not mentioned in this request. When a dynamic property is explicitly mentioned in a POST/PUT/MERGE request and has a null value, the server may choose to preserve its existence as a null dynamic property or remove it from the entry, as the rest of the system should behave as if it was there but had a null value already.

**OData: AtomPub Format**

**Introduction**

OData supports two formats for representing the resources (Collections, Entries, Links, etc) it exposes: the XML-based Atom format and the JSON format. This document describes how OData resources are represented in Atom (plus additional elements defined in AtomPub) and [[OData:JSON]](http://www.odata.org/developers/protocols/json-format) describes the JSON representation. The [content type negotiation](http://www.odata.org/developers/protocols/operations#RepresentationFormatAndContentTypeNegotiation) section of the [[OData:Operations]](http://www.odata.org/developers/protocols/operations) document describes how clients can use standard HTTP content type negotiation to tell an OData service which format it wants to use.

**Table of Contents**

1. [1](http://www.odata.org/developers/protocols/atom-format#Background). Background
2. [2](http://www.odata.org/developers/protocols/atom-format#ATOMRepresentations). ATOM Representations
3. [2.1](http://www.odata.org/developers/protocols/atom-format#PrimitiveTypes). Primitive Types
4. [2.2](http://www.odata.org/developers/protocols/atom-format#ServiceDocuments). Service Documents
5. [2.3](http://www.odata.org/developers/protocols/atom-format#RepresentingCollectionsofEntries). Representing Collections of Entries
6. [2.4](http://www.odata.org/developers/protocols/atom-format#RepresentingEntries). Representing Entries
7. [2.4.1](http://www.odata.org/developers/protocols/atom-format#DeferredContent). Deferred Content
8. [2.4.2](http://www.odata.org/developers/protocols/atom-format#InlineRepresentationofAssociatedEntries). Inline Representation of Associated Entries
9. [2.4.3](http://www.odata.org/developers/protocols/atom-format#RepresentingMediaLinkEntries). Representing Media Link Entries
10. [2.4.4](http://www.odata.org/developers/protocols/atom-format#CustomizingtheRepresentationofanEntry). Customizing the Representation of an Entry
11. [2.5](http://www.odata.org/developers/protocols/atom-format#RepresentingPrimitiveProperties). Representing Primitive Properties
12. [2.6](http://www.odata.org/developers/protocols/atom-format#RepresentingComplexTypesProperties). Representing Complex Types Properties
13. [3](http://www.odata.org/developers/protocols/atom-format#XMLRepresentations). XML Representations
14. [3.1](http://www.odata.org/developers/protocols/atom-format#RepresentingPropertyValues). Representing Property Values
15. [3.1.1](http://www.odata.org/developers/protocols/atom-format#RepresentingPrimitiveProperties). Representing Primitive Properties
16. [3.1.2](http://www.odata.org/developers/protocols/atom-format#RepresentingComplexTypesProperties). Representing Complex Types Properties
17. [3.1.3](http://www.odata.org/developers/protocols/atom-format#RepresentingtheRawvalueofaProperty). Representing the Raw value of a Property
18. [3.2](http://www.odata.org/developers/protocols/atom-format#RepresentingLinks). Representing Links
19. [3.3](http://www.odata.org/developers/protocols/atom-format#RepresentingResultsfromServiceOperations). Representing Results from Service Operations

**1. Background**

As described in Atom [[RFC4287]](http://tools.ietf.org/html/rfc4287), Atom is an XML-based document format that describes Collections of related information known as "feeds". Feeds are composed of a number of items, known as Entries. AtomPub [[RFC5023]](http://tools.ietf.org/html/rfc5023) defines additional format constructs for Entries and Feeds to enable the resources they represent to be easily categorized, grouped, edited and discovered. For the remainder of this document, the term Atom is used to represent the combination of the format/representation rules defined in Atom[[RFC4287]](http://tools.ietf.org/html/rfc4287) and AtomPub [[RF5023]](http://tools.ietf.org/html/rfc5023).

As noted in the OData Basics section of [OData:Core], OData services expose Collections of structured Entries, making Atom a natural fit for representing OData resources. Since Atom does not define how structured data is encoded with feeds, to enable transfers of structured content by OData services, this document defines a set of conventions for representing structured data in an Atom feed.

It should be noted that feeds following the conventions defined in this document are valid AtomPub feeds and can be consumed by feed readers, tools, etc. which are only aware of the Atom standards ([RFC4287] & [RFC5023]), but not the additional conventions defined in this document.

**2. Atom Representations**

The following sections define how resources (Collection, Entries, etc) exposed by an OData service can be represented in requests and responses payloads using the Atom format. For details regarding how to create various request types (Retrieve, Create, etc) see [[OData:Operations]](http://www.odata.org/developers/protocols/operations) .

Through out this section the notation <atom:elementName> is used to refer to the named element in the Atom [[RFC4287]](http://tools.ietf.org/html/rfc4287) specification.

**2.1. Primitive Types**

Values of OData primitive types are represented as values of XML elements/attributes as per the table below. Note: The type system used by OData services is described in full in the [primitive types](http://www.odata.org/developers/protocols/overview#AbstractTypeSystem) section of the [[OData:Core]](http://www.odata.org/developers/protocols/overview) document. In addition to the rules stated in the table, if the value of a primitive type is null, then it is represented as an empty XML element with an **m:null="true"** attribute ("m" identifies the OData metadata namespace).

|  |  |
| --- | --- |
| **Primitive Type** | **Serialization Format in XML Documents** |
| Edm.Binary | Base64 encoded value of an EDM.Binary value. See [[RFC3548]](http://go.microsoft.com/fwlink/?LinkId=90432) . |
| Edm.Boolean | true | false |
| Edm.Byte | Same as the [literal form in [OData:Core]](http://www.odata.org/developers/protocols/overview#AbstractTypeSystem), except without the surrounding |
| Edm.DateTime | Same as the [literal form in [OData:Core]](http://www.odata.org/developers/protocols/overview#AbstractTypeSystem), except without the surrounding datetime'' syntax |
| Edm.Decimal | Same as the [literal form in [OData:Core]](http://www.odata.org/developers/protocols/overview#AbstractTypeSystem), except without the trailing "m" |
| Edm.Double | Same as the [literal form in [OData:Core]](http://www.odata.org/developers/protocols/overview#AbstractTypeSystem) |
| Edm.Guid | Same as the [literal form in [OData:Core]](http://www.odata.org/developers/protocols/overview#AbstractTypeSystem), except without the surrounding guid'' syntax |
| Edm.Int16 | Same as the [literal form in [OData:Core]](http://www.odata.org/developers/protocols/overview#AbstractTypeSystem) |
| Edm.Int32 | Same as the [literal form in [OData:Core]](http://www.odata.org/developers/protocols/overview#AbstractTypeSystem) |
| Edm.Int64 | Same as the [literal form in [OData:Core]](http://www.odata.org/developers/protocols/overview#AbstractTypeSystem) |
| Edm.SByte | Same as the [literal form in [OData:Core]](http://www.odata.org/developers/protocols/overview#AbstractTypeSystem) |
| Edm.Single | Same as the [literal form in [OData:Core]](http://www.odata.org/developers/protocols/overview#AbstractTypeSystem), except without the trailing "f" |
| Edm.String | Same as the [literal form in [OData:Core]](http://www.odata.org/developers/protocols/overview#AbstractTypeSystem), except without surrounding double quotes |
| Edm.Time | Same as the [literal form in [OData:Core]](http://www.odata.org/developers/protocols/overview#AbstractTypeSystem), except without the surrounding time'' syntax |
| Edm.DateTimeOffset | Same as the [literal form in [OData:Core]](http://www.odata.org/developers/protocols/overview#AbstractTypeSystem), except without the surrounding datetimeoffset'' syntax |

**2.2. Service Documents**

As described in [[OData:Core]](http://www.odata.org/developers/protocols/overview), if a service exposes several Collections, then to aid discovery of those Collections by clients it is useful for the service to expose a Service Document which lists the available Collections. Service Documents are described in AtomPub [[RFC5023], section 15](http://tools.ietf.org/html/rfc5023#page-15).

For example, the URI <http://services.odata.org/OData/OData.svc> identifies the Service Document of a sample OData service which exposes a Categories, Products and Suppliers Collection. For convenience, a sample Service Document is shown in the listing below.

<?xml version="1.0" encoding="utf-8" standalone="yes" ?>

<service xml:base="http://services.odata.org/OData/OData.svc/"

xmlns:atom="http://www.w3.org/2005/Atom"

xmlns:app="http://www.w3.org/2007/app"

xmlns="http://www.w3.org/2007/app">

<workspace>

<atom:title>Default</atom:title>

<collection href="Products">

<atom:title>Products</atom:title>

</collection>

<collection href="Categories">

<atom:title>Categories</atom:title>

</collection>

<collection href="Suppliers">

<atom:title>Suppliers</atom:title>

</collection>

</workspace>

</service>

**2.3. Representing Collections of Entries**

[Collections](http://www.odata.org/developers/protocols/terminology#Collection) represent a set of [Entries](http://www.odata.org/developers/protocols/terminology#Entry). In OData, Collections are represented as Atom feeds ([RFC5023] ), with one Atom entry for each Entry within the Collection. For example, a Collection of product category Entries (that could be part of a product catalog) exposed by an OData service, as identified by the URI <http://services.odata.org/OData/OData.svc/Categories>, is represented as shown below. The format of <atom:entry> elements within a feed is described in the [Representing Entries](http://www.odata.org/#RepresentingEntries) section.

**Note**: The "m" and "d" prefixes represent the OData metadata and data namespaces. It is likely the next version of OData will generalize the namespace URI to use an odata.org based URI.

**OData V1:**

<?xml version="1.0" encoding="utf-8" standalone="yes"?>

<feed xml:base=http://services.odata.org/OData/OData.svc/

xmlns:d=http://schemas.microsoft.com/ado/2007/08/dataservices

xmlns:m=http://schemas.microsoft.com/ado/2007/08/dataservices/metadata

xmlns="http://www.w3.org/2005/Atom">

<title type="text">Categories</title>

<id>http://services.odata.org/OData/OData.svc/Categories</id>

<updated>2010-03-10T08:38:14Z</updated>

<link rel="self" title="Categories" href="Categories" />

<entry>

<id>http://services.odata.org/OData/OData.svc/Categories(0)</id>

<title type="text">Food</title>

<updated>2010-03-10T08:38:14Z</updated>

<author>

<name />

</author>

<link rel="edit" title="Category" href="Categories(0)" />

<link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Products"

type="application/atom+xml;type=feed"

title="Products" href="Categories(0)/Products" />

<category term="ODataDemo.Category"

scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />

<content type="application/xml">

<m:properties>

<d:ID m:type="Edm.Int32">0</d:ID>

<d:Name>Food</d:Name>

</m:properties>

</content>

</entry>

<!-- <entry> elements representing additional Categories go here -->

</feed>

In response payloads only, OData v2 supports two pieces of collection-level metadata: an Entry count (the total count of the number of entities in the Collection) and "next links" in the case when a partial listing of the Collection of Entries is being represented.

The Entry count, is only included in the feed returned by an OData service when the request URI includes the $inlinecount System Query Option. In this case, the count information is represented as a <m:count> element with the value of the element being the total number of Entries in the Collection. See the [$inlinecount section](http://www.odata.org/developers/protocols/uri-conventions#InlinecountSystemQueryOption) of the [[OData:URI]](http://www.odata.org/developers/protocols/uri-conventions) document for a description of how the count value is calculated.

In response payloads only, if the server does not include an <entry> element for every Entry in the Collection identified by the request URI (typically for resource conservation reasons), then the returned feed represents a partial listing as defined in AtomPub [[RFC5023] section 10.1](http://tools.ietf.org/html/rfc5023#section-10.1). In this case, the response includes a <link rel="next" href="â€¦"> element to indicate the feed is a partial listing and to provide the URI of the next partial list feed so a client can easily continue obtaining additional entries. Version Note: Partial lists of Entries are supported in OData V2.0 only. For more information on interacting with partial listings, see [Retrieving feeds, entries and service documents](http://www.odata.org/developers/protocols/operations#RetrievingFeedsEntriesAndServiceDocuments) in the [[OData:Operations]](http://www.odata.org/developers/protocols/operations) document.

**OData V2:**

<feed xml:base=<http://services.odata.org/Northwind/Northwind.svc/>

xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"

xmlns:m=http://schemas.microsoft.com/ado/2007/08/dataservices/metadata

xmlns="http://www.w3.org/2005/Atom">

<title type="text">Customers</title>

<id>http://services.odata.org/Northwind/Northwind.svc/Customers</id>

<updated>2010-03-10T09:34:11Z</updated>

<link rel="self" title="Customers" href="Customers" />

**<m:count>91</m:count>**

<entry>

<id>http://services.odata.org/Northwind/Northwind.svc/Customers('ALFKI')</id>

<title type="text" />

<updated>2010-03-10T09:34:11Z</updated>

<author>

<name />

</author>

<link rel="edit" title="Customer" href="Customers('ALFKI')" />

<link rel=http://schemas.microsoft.com/ado/2007/08/dataservices/related/Orders

type="application/atom+xml;type=feed"

title="Orders" href="Customers('ALFKI')/Orders" />

<link rel=http://schemas.microsoft.com/ado/2007/08/dataservices/related/CustomerDemographics

type="application/atom+xml;type=feed"

title="CustomerDemographics" href="Customers('ALFKI')/CustomerDemographics" />

<category term="NorthwindModel.Customer"

scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />

<content type="application/xml">

<m:properties>

<d:CustomerID>ALFKI</d:CustomerID>

<d:CompanyName>Alfreds Futterkiste</d:CompanyName>

<d:ContactName>Maria Anders</d:ContactName>

<d:ContactTitle>Sales Representative</d:ContactTitle>

<d:Address>Obere Str. 57</d:Address>

<d:City>Berlin</d:City>

<d:Region m:null="true" />

<d:PostalCode>12209</d:PostalCode>

<d:Country>Germany</d:Country>

<d:Phone>030-0074321</d:Phone>

<d:Fax>030-0076545</d:Fax>

</m:properties>

</content>

</entry>

<!-- <entry> elements representing additional customers goes here -->

**<link rel="next"**

**href="http://services.odata.org/Northwind/Northwind.svc/Customers?$inlinecount=allpages&$skiptoken='ERNSH'" />**

</feed>

**2.4. Representing Entries**

In OData, Entries are represented as Atom <atom:entry> elements with all the [Properties](http://www.odata.org/developers/protocols/terminology#Property) of the Entry represented as elements within the <m:properties> element which is a direct child of the <atom:content> element. When using an OData v2 server, clients may indicate that they want a subset of the properties by using the Select System Query Option in the request.

If the Entry being represented links to other Entries via [Navigation Properties](http://www.odata.org/developers/protocols/terminology#NavigationProperty) (e.g. a Product is related to a Category), then the Links are represented as <atom:link rel= [http://schemas.microsoft.com/ado/2007/08/dataservices/related/[NavigationPropertyName]](http://schemas.microsoft.com/ado/2007/08/dataservices/related/%5bNavigationPropertyName%5d) href=â€â€¦"/> elements â€" one for each Navigation Property of the Entry.

Metadata describing the Entry being represented can be specified using additional Atom-defined and OData-defined elements/attributes as defined by the following list.

**<atom:id>**

* As per [[RFC4287]](http://tools.ietf.org/html/rfc4287), contains a URI which uniquely identifies the Entry

**<atom:category scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme">**

* This element may be present with a term attribute whose value indicates the Entity Type in the [data model](http://www.odata.org/developers/protocols/overview#AbstractDataModel) of the OData service that describes the Entry represented by the parent <atom:entry> element. An <atom:Entry> must contain at most one such category element with the specified scheme.
* To ensure type fidelity across of Entries sent from server to client (and vice versa), this element must be included if the Entry is part of a type hierarchy and is not the base type in the hierarchy.
* The value of the element is the namespace qualified name of the [Entity Type](http://www.odata.org/developers/protocols/overview#AbstractDataModel) in the [Service Metadata Document](http://www.odata.org/developers/protocols/terminology#ServiceMetadataDocument) that describes the Entry.
* When this element is not present clients and servers should assume that the entry is of the base type of the containing collection, if that information is known. When a collection can contain a number of types within a inheritance hierarchy servers will typically reject requests that contain Entries without type information.

**<atom:link rel="self" href=".." >**

* As per [[RFC4287]](http://tools.ietf.org/html/rfc4287), this element is optional. If included it contains the URI a client should use to retrieve the Entry as described in [[OData:Operations]](http://www.odata.org/developers/protocols/operations).

**<atom:link rel="edit" href=" .. " >**

* As per [[RFC5023]](http://tools.ietf.org/html/rfc5023), this element should be included. If included it contains the URI a client should use to update or delete the Entry as defined in [[OData:Operations]](http://www.odata.org/developers/protocols/operations).

**m:etag attribute**

* An attribute in the OData Metadata Namespace whose value is the [concurrency token](http://www.odata.org/developers/protocols/operations#ConcurrencyControlAndETags) associated with the Entry. The value is formatted as required by the ETag header in [RFC2616].
* This attribute should be present in a response from an OData service when returning a feed (i.e. multiple Entries in a single HTTP response), otherwise the HTTP ETag response header should be used to communicate the concurrency token of the resource returned to a client.

For example, as shown in the following listing, the Category Entry identified by the URI [http://services.odata.org/OData/OData.svc/Categories(0)](http://services.odata.org/OData/OData.svc/Categories%280%29), has two primitive properties (ID & Name) and one Navigation Property named â€œProductsâ€, which identifies a feed of Product Entries related to the Category.

<?xml version="1.0" encoding="utf-8" standalone="yes" ?>

<entry xml:base="http://services.odata.org/OData/OData.svc/"

xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"

xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"

xmlns="http://www.w3.org/2005/Atom">

<id>http://services.odata.org/OData/OData.svc/Categories(0)</id>

<title type="text">Food</title>

<updated>2010-03-10T10:43:51Z</updated>

<author>

<name />

</author>

<link rel="edit" title="Category" href="Categories(0)" />

**<link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Products"**

**type="application/atom+xml;type=feed"**

**title="Products"**

**href="Categories(0)/Products" />**

<category term="ODataDemo.Category"

scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />

**<content type="application/xml">**

**<m:properties>**

**<d:ID m:type="Edm.Int32">0</d:ID>**

**<d:Name>Food</d:Name>**

**</m:properties>**

**</content>**

</entry>

**2.4.1. Deferred Content**

To conserve resources (bandwidth, CPU, and so on), it is generally not a good idea for an OData service to return the full graph of Entries related to the Entry (or Collection of entries) identified in a request URI. For example, an OData service should defer sending related Entries unless the client explicitly asked for them using the [$expand System Query Option](http://www.odata.org/developers/protocols/uri-conventions#ExpandSystemQueryOption) which provides a way for a client to state related entities should be [represented inline](http://www.odata.org/#InlineRepresentationofAssociatedEntries).

As shown in the example in the prior section, by default properties which represent [Links](http://www.odata.org/developers/protocols/terminology#Link) (the "Products" property in the example) are represented as a <link rel="  http://schemas.microsoft.com/ado/2007/08/dataservices/related/[NavigationPropertyName]" href="..." /> element to indicate the service deferred representing the related Entries. If needed, a client can then use the URI in the href attribute in a subsequent retrieve request to obtain the related Entries.

**2.4.2. Inline Representation of Associated Entries**

As described in the [$expand System Query Option section](http://www.odata.org/developers/protocols/uri-conventions#ExpandSystemQueryOption) of the [[OData:URI]](http://www.odata.org/developers/protocols/uri-conventions) document, a request URI may include the $expand query option to explicitly request that a linked to Entry or collection of Entries be serialized inline, rather than deferred.

In this case the related Entry or collection of Entries is represented as the child element of an <m:inline> element as an <atom:feed> or <atom:entry> respectively. For example, a single Category Entry with its related Product Entries serialized inline is represented as shown in the example below.

<?xml version="1.0" encoding="utf-8" standalone="yes"?>

<entry xml:base=http://services.odata.org/OData/OData.svc/"

xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"

xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"

xmlns="http://www.w3.org/2005/Atom">

<id>http://services.odata.org/OData/OData.svc/Categories(0)</id>

<title type="text">Food</title>

<updated>2010-03-11T06:40:04Z</updated>

<author>

<name />

</author>

<link rel="edit" title="Category" href="Categories(0)" />

<link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Products"

type="application/atom+xml;type=feed"

title="Products" href="Categories(0)/Products">

**<m:inline>**

<feed>

<title type="text">Products</title>

<id>http://services.odata.org/OData/OData.svc/Categories(0)/Products</id>

<updated>2010-03-11T06:40:04Z</updated>

<link rel="self" title="Products" href="Categories(0)/Products" />

<entry>

<id>http://services.odata.org/OData/OData.svc/Products(0)</id>

<title type="text">Bread</title>

<summary type="text">Whole grain bread</summary>

<updated>2010-03-11T06:40:04Z</updated>

<author>

<name />

</author>

<link rel="edit" title="Product" href="Products(0)" />

<link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Category"

type="application/atom+xml;type=entry"

title="Category" href="Products(0)/Category" />

<link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Supplier"

type="application/atom+xml;type=entry"

title="Supplier" href="Products(0)/Supplier" />

<category term="ODataDemo.Product"

scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />

<content type="application/xml">

<m:properties>

<d:ID m:type="Edm.Int32">0</d:ID>

<d:ReleaseDate m:type="Edm.DateTime">1992-01-01T00:00:00</d:ReleaseDate>

<d:DiscontinuedDate m:type="Edm.DateTime" m:null="true" />

<d:Rating m:type="Edm.Int32">4</d:Rating>

<d:Price m:type="Edm.Decimal">2.5</d:Price>

</m:properties>

</content>

</entry>

</feed>

**</m:inline>**

</link>

<category term="ODataDemo.Category"

scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />

<content type="application/xml">

<m:properties>

<d:ID m:type="Edm.Int32">0</d:ID>

<d:Name>Food</d:Name>

</m:properties>

</content>

</entry>

**2.4.3. Representing Media Link Entries**

[Media Link Entries (MLE)](http://www.odata.org/developers/protocols/terminology#MediaLinkEntry) are represented in the same way as regular Entries as described in [Representing Entries](http://www.odata.org/#RepresentingEntries); however, they also contain additional metadata per Entry that describes the Media Resource (MR) associated with the Entry and the <m:properties> element becomes a child of the <atom:entry> element. The <m:properties> element is moved out from under the <atom:content> element because in the case of an MLE, the content describes the MR.

This additional MR-specific metadata is represented by the following constructs in the <atom:entry>:

* **<link rel="edit-media" href="{MediaResourceUri}">**
  + As per [[RFC5023] section 9.6](http://tools.ietf.org/html/rfc5023#section-9.6), this element indicates the URI a client should use to update or delete the Media Resource, as described in [[OData:Operations]](http://www.odata.org/developers/protocols/operations).
  + If the ETag of the Media Resource (MR) is independent from that of the Media Link Entry (MLE), then this element should include an m:etag attribute with value equal to the concurrency token of the MR.
* **<atom:content src="{MediaResourceUri}" type="{MimeType}">**
  + For MLE's, this element describes the associated MR. The value of the src attribute is the URI a client should use to retrieve the Media Resource, as described in [[OData:Operations]](http://www.odata.org/developers/protocols/operations). Note: this allows a Media Resource to have independent links for editing and retrieval.
  + The value of the type attribute is the mime type of the MR.

**2.4.4. Customizing the Representation of an Entry**

Services may wish to have more flexibility over how Entries are represented within an <atom:entry> element. For example, a service may want to enable the value of a property of an Entry be represented as the value of one of the standard Atom elements (Title, Summary, etc) or as the value of a custom element within an Entry. OData supports services that need this kind of control over the Atom representations of an Entry.

In general a service may choose to deviate from the conventions defined in Representing Entries section above and represent the value of a Property of an Entry as the value of a standard Atom element (Title, Summary, etc) or as the value of an element in a custom namespace. When a service does this form of customization it breaks the shared assumption between client and server regarding how Entries are encoded within an <atom:entry> element. OData services that deviate from the prescribed encoding should expose a [Service Metadata Document](http://www.odata.org/developers/protocols/terminology#ServiceMetadataDocument) that include Feed Customization annotations which allow a client to discover how the server chose to encode a given Entry within the <atom:entry> element. The remainder of this section describes the set of Feed Customization annotations that may be used.

The following table lists all the Feed Customization annotations defined in OData.

|  |  |
| --- | --- |
| **Feed Customization Annotation** | **Description** |
| FC\_SourcePath | The name of the Property on an Entry which this feed mapping rule applies to.  The format of this parameter is a path expression where property names are separated by a â€˜/â€™ character. For example for a Person Entry with an integer Property â€˜Ageâ€™ and an â€˜Addressâ€™ Complex Type Property, the following are legal values for this property:  - Age  - Address/Street  The following are invalid values for this property:  - Foo (not a defined property)  - Address (doesnâ€™t identify a primitive property)  - Address/Street/ (cannot end with a â€˜/â€™)  - Empty string or null  - /Address/Street (cannot start with a â€˜/â€™) |
| FC\_TargetPath | The name of the element to map the property identified by the FC\_SourcePath annotation to. If no FC\_SourcePath annotation is specified, then this annotation must be on a <Property> element in the Service Metadata Document which itself identifies the source Property.  The format of this annotation is a path expression where nested elements are separated by a â€˜/â€™ and attributes are specified by a â€˜@â€™ symbol.  For example, to map to a property value to a custom XML element or attribute:  To map the source Property to a custom <Category> element which is a direct child of the <atom:entry> element use:  - FC\_TargetPath="Category"  To map the source Property to an attribute named Term on a custom <Category> element which is a direct child of the <atom:entry> element:  - FC\_TargetPath="Category/@Term"  To map the source Property to a child element of the custom <Category> element named <Foo>, where <Category> is a direct child of the <atom:entry> element:  - FC\_TargetPath="Category/Foo"  This annotation also supports mapping to Atom-defined elements. This following list describes the values used to map to Atom-defined elements:  Format: value : mapped to atom element  â€¢ "SyndicationAuthorEmail" : <author><email>  â€¢ "SyndicationAuthorName" : <author><name>  â€¢ "SyndicationAuthorUri" : <author><uri>  â€¢ "SyndicationPublished" : <published>  â€¢ "SyndicationRights" : <rights>  â€¢ "SyndicationSummary" : <summary>  â€¢ "SyndicationTitle" : <title>  Note: This annotation does not support mapping to multiple instances of an element, so two overlapping path definitions imply the same element instances. For example: the target path values â€œa/b/câ€ and â€œa/d/fâ€ imply the following XML structure: <a><b> <c/></b><d><f/></d> |
| FC\_ContentKind | The mime type of the source Property |
| FC\_NsPrefix | When the source Property is being mapped to an element that is not in the Atom namespace, the value of this annotation specifies the namespace prefix to use the Atom document for target element. |
| FC\_NsUri | When a property is being mapped to an element that is not in the Atom namespace, this annotation is used to specify the namespace for the target element. The FC\_NSPrefix annotation specifies the namespace prefix for the namespace specified by this annotation. |
| FC\_KeepInContent | The value of this property is true if the property being mapped should appear both in its â€œmapped toâ€ location as well as within the content section of the <entry>.  If the value of this property is false, then the property being mapped should only appear in its new â€œmapped toâ€ location in the <entry> and NOT also in the content section. This value (false) is only supported in OData v2.0 and requires that any response formatted in this way return a DataServiceVersion:2.0 response header. |

For example, the sample Service Metadata Document at http://services.odata.org/OData/OData.svc/$metadata states the Name Property of the Category Entry is represented as the value of the <atom:title> element. Since FC\_KeepInContent is equals true for this mapping the value is duplicated within the <m:properties> element.

**Snippet from Service Metadata Document**

<EntityType Name="Category">

<Key>

<PropertyRef Name="ID" />

</Key>

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

<Property Name="Name" Type="Edm.String"

Nullable="true"

**m:FC\_TargetPath="SyndicationTitle"**

**m:FC\_ContentKind="text"**

**m:FC\_KeepInContent="true"** />

<NavigationProperty Name="Products"

Relationship="ODataDemo.Product\_Category\_Category\_Products"

FromRole="Category\_Products" ToRole="Product\_Category" />

</EntityType>

**Category representation in Atom**

(see http://services.odata.org/OData/OData.svc/Categories(0) )

<?xml version="1.0" encoding="utf-8" standalone="yes"?>

<entry xml:base="http://services.odata.org/(S(S\_ODATA\_RO))/OData/OData.svc/" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom">

<id>http://services.odata.org/(S(S\_ODATA\_RO))/OData/OData.svc/Categories(0)</id>

**<title type="text">Food</title>**

<updated>2010-03-11T22:28:51Z</updated>

<author>

<name />

</author>

<link rel="edit" title="Category" href="Categories(0)" />

<link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Products"

type="application/atom+xml;type=feed"

title="Products"

href="Categories(0)/Products" />

<category term="ODataDemo.Category"

scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />

<content type="application/xml">

<m:properties>

<d:ID m:type="Edm.Int32">0</d:ID>

**<d:Name>Food</d:Name>**

</m:properties>

</content>

</entry>

If the Category Entry in the example above was part of a type hierarchy and the Name property was defined on the type Category derives from, then the mapping from Name to <atom:title> is still possible, however the placement of the annotation changes to be on the <EntityType> element in the Service Metadata Document as shown in the following listing.

<EntityType Name="Category" **m:FC\_TargetPath="SyndicationTitle"**

**m:FC\_SourcePath="Name"**

**m:FC\_ContentKind="text"**

**m:FC\_KeepInContent="true"**

BaseType:"Sample.BaseCategory">

<Key>

<PropertyRef Name="ID" />

</Key>

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

<NavigationProperty Name="Products"

Relationship="ODataDemo.Product\_Category\_Category\_Products"

FromRole="Category\_Products" ToRole="Product\_Category" />

</EntityType>

Instead of mapping the Name property to an Atom-defined element, a Property may be mapped to a custom element in any namespace. For example, the following listing maps the Street Property of the Address Complex Type on a Supplier entry to a custom <c:Road> element.

Snippet from Service Metadata Document

<EntityType Name="Supplier">

<Key>

<PropertyRef Name="ID" />

</Key>

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

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

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

**m:FC\_TargetPath="Road"**

**m:FC\_SourcePath="Street"**

**m:FC\_KeepInContent="true"**

**m:FC\_NsUri="http://customUri"**

**m:FC\_NsPrefix="c"/>**

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

<NavigationProperty Name="Products"

Relationship="ODataDemo.Product\_Supplier\_Supplier\_Products"

FromRole="Supplier\_Products"

ToRole="Product\_Supplier" />

</EntityType>

<ComplexType Name="Address">

<Property Name="Street" Type="Edm.String" Nullable="true" />

<Property Name="City" Type="Edm.String" Nullable="true" />

<Property Name="State" Type="Edm.String" Nullable="true" />

<Property Name="ZipCode" Type="Edm.String" Nullable="true" />

<Property Name="Country" Type="Edm.String" Nullable="true" />

</ComplexType>

Supplier representation in Atom

<?xml version="1.0" encoding="utf-8" standalone="yes"?>

<entry xml:base="http://services.odata.org/(S(S\_ODATA\_RO))/OData/OData.svc/" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" m:etag="W/"0"" xmlns="http://www.w3.org/2005/Atom"

Xmlns:c="http://customUri">

<id>http://services.odata.org/(S(S\_ODATA\_RO))/OData/OData.svc/Suppliers(0)</id>

<title type="text">Exotic Liquids</title>

<updated>2010-03-11T22:47:21Z</updated>

<author>

<name />

</author>

<link rel="edit" title="Supplier" href="Suppliers(0)" />

<link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Products" type="application/atom+xml;type=feed" title="Products" href="Suppliers(0)/Products" />

<category term="ODataDemo.Supplier" scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />

**<c:Road>NE 228th</c:Road>**

<content type="application/xml">

<m:properties>

<d:ID m:type="Edm.Int32">0</d:ID>

<d:Name>Exotic Liquids</d:Name>

<d:Concurrency m:type="Edm.Int32">0</d:Concurrency>

<d:Address m:type="ODataDemo.Address">

<d:Street>NE 228th</d:Street>

<d:City>Sammamish</d:City>

<d:State>WA</d:State>

<d:ZipCode>98074</d:ZipCode>

<d:Country>USA</d:Country>

</d:Address>

</m:properties>

</content>

</entry>

**2.5. Representing Primitive Properties**

When represented in a request/response payload as part of an Entry, Complex Type or a standalone construct in a request payload, primitive properties are represented in Atom as child elements of an <m:properties> element (see Representing Entities) with the name equal to the element equal to the property and value of the element set to the primitive type value formatted as described by the table in the [Primitive Types](http://www.odata.org/developers/protocols/atom-format#PrimitiveTypes) section above.

See [section 3.1](http://www.odata.org/developers/protocols/atom-format#RepresentingPropertyValues) for the format of Properties when they are represented independently from their defining Entry (i.e. outside the context of the defining Entry).

**2.6. Representing Complex Type Properties**

When represented as a property of an Entry or Complex Type (also within an Entry) in a request/response payload, a property whose type is a [complex type](http://www.odata.org/developers/protocols/overview#AbstractDataModel) is represented as an XML element with each property of the complex type represented as a direct child element (as described in the prior section for primitive properties). For example, the Address Complex Type of a Supplier Entry is shown in the example below.

See [section 3.1.2](http://www.odata.org/developers/protocols/atom-format#RepresentingComplexTypesProperties) for the format of Complex Type Properties when they are represented independently from their defining Entry (i.e. outside the context of the defining Entry).

**3. XML Representations**

The smallest unit of information that can be represented in Atom is an Entry. In several application scenarios it is practical to be able to identify a specific piece of information within the Entry, particularly when the entry is mapped to an application-level construct on the server. For example, a Presentation Entry may have a Property that is the abstract describing the presentation.

To enable this scenario, as described in the [Resource Path](http://www.odata.org/developers/protocols/uri-conventions#ResourcePath) section in the [[OData:URI]](http://www.odata.org/developers/protocols/uri-conventions) document, OData supports directly addressing a Property of an Entry. Since there is not a natural mapping of a single Property (outside of the context of the defining Entry) to an Atom representation, OData represents the constituent parts of an Entry using a simple XML representation, which is described by the following subsections.

**3.1. Representing Property Values**

As described in the [Resource Path](http://www.odata.org/developers/protocols/uri-conventions#ResourcePath) section in the [[OData:URI]](http://www.odata.org/developers/protocols/uri-conventions) document, OData supports directly addressing a Property of an Entry. The following subsections describe how each type of Property is represented in XML.

**3.1.1. Representing Primitive Properties**

When a primitive Property is represented as a standalone construct in a response from an OData service (such as when a retrieve request is made to a URI that identifies a single Property) it is represented as a single XML element where the element name equals the name of the Property and the value of the element is the value Property value. For example, the response payload to a retrieve request that identifies the Name property of a Category Entry is represented as shown below.

<?xml version="1.0" encoding="utf-8" standalone="yes"?>

<Name xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices">Bread</Name>

**3.1.2. Representing Complex Types Properties**

When a Property whose type is a [complex type](http://www.odata.org/developers/protocols/overview#AbstractDataModel) is represented as a standalone construct, such as when a retrieve request is made to a URI that identifies a single Property, it uses a single XML element with the name of the Property and child elements for each Property defined on the complex type (formatted as per this section of the prior â€œRepresenting Primitive Propertiesâ€ section). For example, the response payload to a retrieve request that identifies the Name property of a Category Entry is represented as shown below.

<?xml version="1.0" encoding="utf-8" standalone="yes"?>

<Address p1:type="ODataDemo.Address"

xmlns:p1="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"

xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices">

<Street>NE 228th</Street>

<City>Sammamish</City>

<State>WA</State>

<ZipCode>98074</ZipCode>

<Country>USA</Country>

</Address>

**3.1.3. Representing the Raw value of a Property**

OData services may support [addressing the â€œrawâ€ value of a primitive Property](http://www.odata.org/developers/protocols/uri-conventions#AddressingEntries) (see the description of the $value URI segment). In this case, the value is returned using the format (aka mime type) the OData service deems to be the â€œrawâ€ format for the Property. For example, the HTTP response from the sample OData service when retrieving the Name string Property of a Category Entry is shown in the example below.

HTTP/1.1 200 OK

DataServiceVersion: 1.0;

Content-Type: text/plain;charset=utf-8

Food

**3.2. Representing Links**

A Link (or collection of Links) represents an associated Entry (or collection of associated Entries). As described in [[OData:Operations]](http://www.odata.org/developers/protocols/operations) Links can be retrieved and modified to change the associations between Entries. A single link is represented as a <d:uri> element with the value of the element set to the URI that identifies the Link. A collection of links is represented as a <d:links> element with zero or more child <d:uri> elements, where each <d:uri> element represents a Link.

Note: in the paragraph above, "d" represents the [OData Data Namespace](http://www.odata.org/developers/protocols/terminology#ODataDataNamespace).

For example, a link with multiplicity 1 (ex. Product is related to a *single* Supplier) would be represented in XML in a response as:

<?xml version="1.0" encoding="utf-8" standalone="yes"?>

<uri xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices">http://services.odata.org/OData/OData.svc/Suppliers(1)</uri>

For example, a link with multiplicity greater than 1 (e.g. Supplier is related to *many* products) would be represented in a response as:

<?xml version="1.0" encoding="utf-8" standalone="yes"?>

<links xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices">

<uri>http://services.odata.org/OData/OData.svc/Products(1)</uri>

<uri>http://services.odata.org/OData/OData.svc/Products(2)</uri>

<uri>http://services.odata.org/OData/OData.svc/Products(3)</uri>

<uri>http://services.odata.org/OData/OData.svc/Products(4)</uri>

<uri>http://services.odata.org/OData/OData.svc/Products(5)</uri>

<uri>http://services.odata.org/OData/OData.svc/Products(6)</uri>

</links>

**3.3. Representing Results from Service Operations**

As described in [[OData:Operations]](http://www.odata.org/developers/protocols/operations) OData services may expose custom behaviors via Service Operations, which may accept input parameters identified by the request URI (as described in [[OData:URI]](http://www.odata.org/developers/protocols/uri-conventions)). This section specifies how the results of a Service Operation are formatted using Atom or XML. Service operations support returning:

* A single primitive value or collection of primitive values
* A single complex type or collection of complex types
* A single Entry or collection of Entries

If a Service Operation returns a collection of Entries or a single Entry, then the Entries (or Entry) are formatted in Atom as described in the [Representing Collections of Entries](http://www.odata.org/#RepresentingCollectionsofEntries) or [Representing an Entry](http://www.odata.org/#RepresentingEntries) section.

If a Service Operation returns a single primitive value, then it is formatted in XML as per the [Representing Primitive Properties](http://www.odata.org/RepresentingPrimitiveProperties) section. Likewise, if a Service Operation returns a single Complex Type value, then it is represented in XML as per the [Representing Complex Type Properties](http://www.odata.org/#RepresentingComplexTypesProperties) section.

If a Service Operation returns a collection of primitive values, then the primitives are returned as the value of <m:element> elements which are children of the root element. The name of the root element should be equal to the name of the Service Operation that was invoked. For example a response payload from a "GetInts" Service Operation that returns a collection of integer values would be formatted as shown in the following listing.

<GetInts xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">

<element>0</element>

<element>1</element>

<element>2</element>

</GetInts>

If a Service Operation returns a collection of complex types, then the collection is represented in XML by a root element whose name is equal to the name of the Service Operation and includes one child <m:element> element for each Complex Type value in the collection returned from the Service Operation. The Properties of each Complex Type are represented as per the [Representing Complex Type Properties](http://www.odata.org/#RepresentingComplexTypesProperties) section. Finally, each <m:element> element should include an m:type attribute that lists the namespace qualified name of the Complex Type. For example, response payload from a "GetAddresses" Service Operation that returns a collection of Address Complex Type values would be represented as show in the following listing.

<GetAddressses xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">

<element type="SampleNamespace.Address">

<street>123 Contoso Ave</street>

<city>London</street>

</element>

<!-- additional Addresses returned go here -->

</GetAddresses>