OData Version 4.0 Part 1: Protocol

Committee Specification Draft 01

26 April 2013

Specification URIs

This version:

http://docs.oasis-open.org/odata/odata/v4.0/csd01/part1-protocol/odata-v4.0-csd01-part1-protocol.doc (Authoritative)
http://docs.oasis-open.org/odata/odata/v4.0/csd01/part1-protocol/odata-v4.0-csd01-part1-protocol.html
http://docs.oasis-open.org/odata/odata/v4.0/csd01/part1-protocol/odata-v4.0-csd01-part1-protocol.pdf

Previous version:

N/A

Latest version:

http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part1-protocol.doc (Authoritative)
http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part1-protocol.html
http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part1-protocol.pdf

Technical Committee:

OASIS Open Data Protocol (OData) TC

Chairs:

Barbara Hartel (barbara.hartel@sap.com), SAP AG
Ram Jeyaraman (Ram.Jeyaraman@microsoft.com), Microsoft

Editors:

Michael Pizzo (mikep@microsoft.com), Microsoft
Ralf Handl (ralf.handl@sap.com), SAP AG
Martin Zurmuehl (martin.zurmuehl@sap.com), SAP AG

Additional artifacts:

This prose specification is one component of a Work Product which consists of:

- *OData Version 4.0 Part 1: Protocol* (this document). http://docs.oasis-open.org/odata/odata/v4.0/csd01/part1-protocol/odata-v4.0-csd01-part1-protocol.html
- *OData Version 4.0 Part 2: URL Conventions*. http://docs.oasis-open.org/odata/odata/v4.0/csd01/part2-url-conventions/odata-v4.0-csd01-part2-url-conventions.html
- *OData Version 4.0 Part 3: Common Schema Definition Language (CSDL)*. http://docs.oasis-open.org/odata/odata/v4.0/csd01/part3-csdl/odata-v4.0-csd01-part3-csdl.html
- ABNF components: *OData ABNF Construction Rules Version 4.0* and *OData ABNF Test Cases.* http://docs.oasis-open.org/odata/odata/v4.0/csd01/abnf/
- Vocabulary components: *OData Core Vocabulary* and *OData Measures Vocabulary*. http://docs.oasis-open.org/odata/odata/v4.0/csd01/vocabularies/
- XML schemas: OData EDMX XML Schema and OData EDM XML Schema. http://docs.oasis-open.org/odata/odata/v4.0/csd01/schemas/
- OData Metadata Service Entity Model: http://docs.oasis-open.org/odata/odata/v4.0/csd01/models/MetadataService.edmx

Related work:

This specification is related to:

- *OData Atom Format Version 4.0*. Latest version. http://docs.oasis-open.org/odata/odata-atom-format/v4.0/odata-atom-format-v4.0.html.
- *OData JSON Format Version 4.0*. Latest version. http://docs.oasis-open.org/odata/odata-json-format/v4.0/odata-json-format-v4.0.html.

Declared XML namespaces:

- http://docs.oasis-open.org/odata/ns/edmx
- http://docs.oasis-open.org/odata/ns/edm

Abstract:

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 an Entity Data Model (EDM), to be published and edited by Web clients using simple HTTP messages. This document defines the core semantics and facilities of the protocol.

Status:

This document was last revised or approved by the OASIS Open Data Protocol (OData) TC on the above date. The level of approval is also listed above. Check the "Latest version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committees/odata/.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/odata/ipr.php).

Citation format:

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

[OData-Part1]

OData Version 4.0 Part 1: Protocol. 26 April 2013. OASIS Committee Specification Draft 01. http://docs.oasis-open.org/odata/odata/v4.0/csd01/part1-protocol/odata-v4.0-csd01-part1-protocol.html.

Notices

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

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

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

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

The name "OASIS" is a trademark of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see http://www.oasis-open.org/policies-guidelines/trademark for above guidance.

Table of Contents

1 Introduction

All text is normative unless otherwise labeled.

1.1 Terminology

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in **[RFC2119]**.

1.2 Normative References

[OData-ABNF] *OData ABNF Construction Rules Version 4.0*.
See link in "Additional artifacts" section on cover page.

[OData-Atom] *OData ATOM Format Version 4.0*.
See link in "Related work" section on cover page.

[OData-CSDL] *OData Version 4.0 Part 3: Common Schema Definition Language (CSDL)*.
See link in "Additional artifacts" section on cover page.

[OData-JSON] *OData JSON Format Version 4.0.*
See link in "Related work" section on cover page.

[OData-URL] *OData Version 4.0 Part 2: URL Conventions*.
See link in "Additional artifacts" section on cover page.

[OData-VocCore] *OData Core Vocabulary.*
See link in "Additional artifacts" section on cover page.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November, 1996. http://www.ietf.org/rfc/rfc2046.txt.

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

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1”, RFC 2616, June 1999. http://www.ietf.org/rfc/rfc2616.txt.

[RFC5023] Gregorio, J., Ed., and B. de hOra, Ed., “The Atom Publishing Protocol.”, RFC 5023, October 2007. http://tools.ietf.org/html/rfc5023.

[RFC5789] Dusseault, L., and J. Snell, “Patch Method for HTTP”, RFC 5789, March 2010. http://tools.ietf.org/html/rfc5789.

[HTTP-Prefer] Snell, J., "Prefer Header for HTTP", draft-snell-http-prefer-18, January 2013. https://datatracker.ietf.org/doc/draft-snell-http-prefer/.

[HTTP-Message] Fielding, R., Ed. and J. Reschke, Ed., “Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing” draft-ietf-httpbis-p1-messaging-22, 23 February 2013. https://datatracker.ietf.org/doc/draft-ietf-httpbis-p1-messaging/.

[HTTP-Semantics] Fielding, R., Ed. and J. Reschke, Ed., “Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content” draft-ietf-httpbis-p2-semantics-22, 23 February 2013 https://datatracker.ietf.org/doc/draft-ietf-httpbis-p2-semantics/.

1.3 Examples

Some sections of this specification are illustrated with non-normative example OData request and response payloads. However, the text of this specification provides the definition of conformance.

All code examples in this document are non-normative.

2 Overview

The OData Protocol is an application-level protocol for interacting with data via RESTful web services. The protocol supports the description of data models and the editing and querying of data according to those models. It provides facilities for:

- Metadata: a machine-readable description of the data model exposed by a particular data provider.
- Data: sets of data entities and the relationships between them.
- Querying: requesting that the service perform a set of filtering and other transformations to its data, then return the results.
- Editing: creating, updating, and deleting data.
- Operations: invoking custom logic
- Vocabularies: attaching custom semantics

The OData Protocol is different from other REST-based web service approaches in that it provides a uniform way to describe both the data and the data model. This improves semantic interoperability between systems and allows an ecosystem to emerge.

Towards that end, the OData Protocol follows these design principles:

- Prefer mechanisms that work on a variety of data stores. In particular, do not assume a relational data model.
- Extensibility is important. Services should be able to support extended functionality without breaking clients unaware of those extensions.
- Follow REST principles unless there is a good and specific reason not to.
- OData should build incrementally. A very basic, compliant service should be easy to build, with additional work necessary only to support additional capabilities.
- Keep it simple. Address the common cases and provide extensibility where necessary.

3 Data Model

This section provides a high-level description of the *Entity Data Model (EDM)*: the abstract data model that MUST be used to describe the data exposed by an OData service. An OData Metadata Document is a representation of a service's data model exposed for client consumption.

The central concepts in the EDM are entities, relationships, entity sets, operations, and containers.

Entities are instances of entity types (e.g. `Customer`, `Employee`, etc.).

Entity types are nominal structured types with a key. They define the named properties of an entity and MAY define relationships with, or derived by single inheritance from, other entity types.

The *entity key* of an entity type is formed from a subset of primitive properties (e.g. `CustomerId`, `OrderId`, `LineId`, etc.) of the entity type

Complex types are keyless nominal structured types consisting of a set of properties. These are value types that lack identity. Complex types are commonly used as property values in an entity or as parameters to operations.

Properties declared as part of a structured type's definition are called *declared properties*. Instances of structured types MAY contain additional undeclared *dynamic properties*. A dynamic property MUST NOT have the same name as a declared property. Entity types which allow clients to persist additional undeclared properties are called *open entity types*.

Relationships from one entity to another are represented as *navigation properties.* Navigation properties are generally defined as part of an entity type, but can also appear on entity instances as undeclared *dynamic navigation properties*. Each relationship has a cardinality.

Enumeration types are named primitive types whose values are named constants with underlying integer values.

Type definitions are named primitive types that MAY have fixed facet values such as maximum length or precision. Type definitions can be used in place of primitive typed properties, for example, within property definitions.

Entity containers and entity sets describe how entities are grouped.

Entity sets are named collections of entities (e.g. `Customers` is an entity set containing `Customer` entities). An entity's key uniquely identifies the entity within an entity set. An entity can be a member of at most one entity set. Entity sets provide entry points into the data model.

Operations allow the execution of custom logic on parts of a data model. *Functions* are operations that do not have side effects and can be further composed, for example, with additional filter operations, functions or an action. *Actions* are operations that allow side effects, such as data modification, and MUST NOT be further composed in order to avoid non-deterministic behavior.

Actions and functions are global to the service and MAY be bound to a type, enabling them to be called as members of an instance of that type. *Action imports* and *function imports* enable actions and functions to be called from an entity container.

Named entities are singleton entities that can be referenced from an entity container.

Named *entity containers* aggregate entity sets, named entities, function imports and action imports.

An OData *resource* is anything in the model that can be addressed (an entity set, entity, property, or operation).

Refer to **[OData-CSDL]** for more information on the OData entity data model.

3.1 Annotations

Model and instance elements can be decorated with *Annotations*.

Annotations can be used to specify an individual fact about an element, such as whether it is read-only, or to define a common concept, such as a person or a movie.

Applied *annotations* consist of a *term* (the namespace-qualified name of the annotation being applied), a *target* (the model or instance element to which the term is applied), and a *value*. The value may be a static value, or an expression that may contain a path to one or more properties of an annotated entity.

Annotation terms are defined in metadata and have a name and a type.

A set of related terms in a common namespace comprises a *Vocabulary*.

4 Service Model

OData services are defined using a common data model. The service advertises its concrete data model in a machine-readable form, allowing generic clients to interact with the service in a well-defined way.

An OData service exposes two well-defined resources that describe its data model; a service document and a metadata document.

The *service document* lists entity sets, functions, and named entities that can be retrieved. Clients can use the service document to navigate the model in a hypermedia-driven fashion.

The *metadata document* describes the types, sets, functions and actions understood by the OData service. Clients can use the metadata document to understand how to query and interact with entities in the service.

In addition to these two “fixed” resources an OData service consists of dynamic resources. The URLs for many of these resources can be computed from the information in the metadata document.

See Requesting Data and Data Modification for details.

4.1 Entity Ids and Resource References

Whereas entities in the data model are uniquely identified by their key values within an entity set, entity instances within a payload are identified by a durable, opaque, globally unique *entity-id*. The entity-id MUST be an IRI as defined in **Error! Reference source not found.**. Services are strongly encouraged to use URLs for entity-ids, but consumers MUST NOT assume this IRI can be used to locate the entity, nor assume any semantics from its structure.

Resource references refer to any addressable resource, such as an entity or the property of an entity. *Entity references* are resource references that refer to an entity using the entity's entity-id.

5 Versioning

Versioning enables clients and services to evolve independently. OData defines semantics for both protocol and data model versioning.

5.1 Protocol Versioning

OData requests and responses SHOULD be versioned according to the `OData-Version` header.

An OData client SHOULD use the `OData-MaxVersion` header in order to specify the maximum acceptable response version. The service SHOULD respond with the maximum version it supports that is less than or equal to the requested `OData-MaxVersion`.

`OData-Version` and `OData-MaxVersion` header fields MUST be of the following form:

```
majorversionnumber "." minorversionnumber
```

This version of the specification defines data service version value `4.0`.

5.2 Model Versioning

The Data Model exposed by an OData Service defines a contract between the OData service and its clients. Services MAY extend their model only to the degree that it does not break existing clients. Breaking changes, such as removing or changing the type of existing properties, require that a new service version is provided at a different service root URL for the new model.

The following Data Model additions are considered safe and do not require services to version their entry point.

- Adding a property that is nullable or has a default value and does not conflict with an existing dynamic property
- Adding a navigation property that is nullable or collection-valued and does not conflict with an existing dynamic navigation property
- Adding a new entity type to the model that does not derive from any entity types exposed by existing entity sets, or returned by existing functions or actions
- Adding a new complex type to the model that does not derive from a complex type used within entity types of existing entity sets, or returned by existing functions or actions
- Adding a new entity set to an entity container
- Adding a new named entity to an entity container
- Adding an action, a function, an action import, or function import
- Adding an action or function parameter that is nullable or has a default value
- Adding a type definition or enumeration
- Adding any annotation to a model element that does not need to be understood by the client in order to correctly interact with the service

Clients SHOULD be prepared for services to make such incremental changes to their model. In particular, clients should be prepared to receive properties not previously defined by the service.

6 Extensibility

The OData protocol supports both user- and version-driven extensibility through a combination of versioning, convention, and explicit extension points.

6.1 Query Option Extensibility

Query options within the request URL can control how a particular request is processed by the service.

OData-defined system query options are prefixed with "`$`". Services MAY support additional query options not defined in the OData specification, but they MUST NOT begin with the "`$`" or "`@`" character.

OData services SHOULD NOT require any query options to be specified in a request. Services SHOULD fail any request that contains query options that they not understand and MUST fail any request that contains unsupported OData query options defined in the version of this specification supported by the service.

6.2 Payload Extensibility

OData supports extensibility in the payload, according to the specific format.

Regardless of the format, additional content MUST NOT be present if it needs to be understood by the receiver in order to correctly interpret the payload according to the specified `OData-Version` header. Thus, clients and services MUST be prepared to handle or safely ignore any content not specifically defined in the version of the payload specified by the `OData-Version` header.

6.3 Action/Function Extensibility

Actions and Functions extend the set of operations that can be performed on or with a service or resource. Actions MAY have side-effects. For example, Actions can be used to modify data or to invoke custom operations. Functions MUST NOT have side-effects. Functions can be invoked from a URL that addresses a resource or within an expression to a `$filter` or `$orderby` system query option.

Fully qualified action and function names MUST include a namespace prefix. The `odata` and `geo` namespaces are reserved for the use of this specification.

An OData service MUST fail any request that contains actions or functions that it does not understand.

6.4 Vocabulary Extensibility

Vocabularies provide the ability to annotate metadata as well as instance data, and define a powerful extensibility point for OData.

Metadata annotations can be used to define additional characteristics or capabilities of a metadata element, such as a service, entity type, property, function, action or parameter. For example, a metadata annotation could define ranges of valid values for a particular property.

Instance annotations can be used to define additional information associated with a particular result, entity, property, or error; for example whether a property is read-only for a particular instance.

Where annotations apply across all instances of a type, services are encouraged to specify the annotation in metadata rather than repeating in each instance of the payload. Where the same annotation is defined at both the metadata and instance level, the instance-level annotation overrides the one specified at the metadata level.

A service MUST NOT require the client to understand custom annotations in order to accurately interpret a response.

OData defines a `Core` vocabulary with a set of basic terms describing behavioral aspects along with terms that can be used in defining other vocabularies, see **[OData-VocCore]**.

6.5 Header Field Extensibility

OData defines semantics around certain HTTP request and response headers. Services that support a version of OData MUST understand and comply with the headers defined by that version. Compliance means either honoring the semantics of the header field or failing the request.

Individual services MAY define custom headers. These headers MUST NOT begin with `OData`. Custom headers SHOULD be optional when making requests to the service. A service MUST NOT require the client to understand custom headers to accurately interpret the response.

6.6 Format Extensibility

An OData service MUST support **[OData-JSON]** or **[OData-Atom]**, and MAY support additional formats for both request and response bodies.

7 Formats

The client MAY request a particular response format through the `Accept` header, as specified in **[RFC2616]**, or through the $format System Query Option.

In the case that both the `Accept` header and the `$format` query option are specified on a request, the value specified in the `$format` query option MUST be used.

If the service does not support the requested format, it SHOULD reply with a `406 Not Acceptable` error response.

See the format specifications (**[OData-JSON]**, **[OData-Atom]**) for details.

8 Header Fields

OData defines semantics around the following request and response headers. Additional headers MAY be specified, but have no unique semantics defined in OData.

8.1 Common Headers

The `OData-Version` and `Content-Type` headers are common between OData requests and responses.

8.1.1 Header `Content-Type`

As specified in **[RFC2616],** the format of an individual request or response body MUST be specified in the `Content-Type` header of the request or response.

See the format-specific specifications (**[OData-JSON]**, **[OData-Atom]**) for details.

8.1.2 Header `Content-Length`

As specified in **[RFC2616],** a request or response SHOULD include a `Content-Length` header when the message's length can be determined prior to being transferred. OData does not add any additional requirements over HTTP for writing `Content-Length`.

8.1.3 Header `OData-Version`

OData clients SHOULD use the `OData-Version` header on a request to specify the version of the protocol used to generate the request.

If present on a request, the service MUST interpret the request according to the rules defined in the specified version of the protocol, or fail the request with a `4xx` response code.

If not specified in a request, the service MUST assume the request is generated using the lesser of the `OData-MaxVersion`, if specified, and the maximum version of the protocol that the service understands.

OData services MUST include the `OData-Version` header on a response to specify the version of the protocol used to generate the response. The client MUST interpret the response according to the rules defined in the specified version of the protocol.

For more details see Versioning.

8.2 Request Headers

In addition to the Common Headers, the client MAY specify any combination of the following request headers.

8.2.1 Header `Accept`

As specified in **[RFC2616]**, the client MAY specify the set of accepted formats through the use of the `Accept` Header.

If the media type specified in the `Accept` Header includes a `charset` format parameter and the request also contains an `Accept-Charset` header, then the `Accept-Charset` header MUST be used.

If the media type specified in the `Accept` Header does not include a `charset` format parameter, then the `Content-Type` header of the response MUST NOT contain a `charset` format parameter.

8.2.2 Header Accept-Charset

As specified in **[RFC2616]**, the client MAY specify the set of accepted character sets through the use of the `Accept-Charset` Header.

8.2.3 Header `If-Match`

As specified in **[RFC2616]**, a client MAY include an `If-Match` header in a request to `GET`, `PUT`, `PATCH` or `DELETE`. The value of the `If-Match` request header MUST be an ETag value previously retrieved for the entity, or `"*"` to match any value.

If an entity set is annotated with the term `Core.OptimisticConcurrencyControl` (see **[OData-VocCore]**) and the client does not specify an `If-Match` request header in a Data Modification Request or Action Request, the service MUST respond with a `428 Precondition Required` for this case and MUST ensure that no observable change occurs as a result of the request.

If specified, the request MUST only be processed if the specified value matches the current ETag value of the target entity. If the value does not match the current ETag value of the entity for a Data Modification Request or Action Request, the service MUST respond with `412 Precondition Failed` and MUST ensure that no observable change occurs as a result of the request. In the case of an upsert, if the addressed entity does not exist the provided ETag value is considered not to match.

The client MAY include an `If-Match` header in a `PUT` or `PATCH` request in order to ensure that the request is handled as an update and not an upsert.

8.2.4 Header `If-None-Match`

As specified in **[RFC2616]**, a client MAY include an `If-None-Match` header in a request to `GET`, `PUT`, `PATCH` or `DELETE`. The value of the `If-None-Match` request header MUST be an ETag value previously retrieved for the entity, or `"*"`.

If specified, the request MUST only be processed if the specified value does not match the current ETag value of the entity. If the value does match the current ETag value of the entity, then for a `GET` request, the service SHOULD respond with `304 Not Modified`, and for a Data Modification Request or Action Request, the service MUST respond with `412 Precondition Failed` and MUST ensure that no observable change occurs as a result of the request.

The client MAY include an `If-None-Match` header with a value of `"*"` in a `PUT` or `PATCH` request in order to ensure that the upsert request is handled as an insert and not an update.

8.2.5 Header `OData-MaxVersion`

Clients SHOULD specify an `OData-MaxVersion` request header.

If specified, the service MUST generate a response with an `OData-Version` less than or equal to the specified `OData-MaxVersion`.

If `OData-MaxVersion` is not specified, then the service SHOULD interpret the request as having an `OData-MaxVersion` equal to the maximum version supported by the service.

For more details see Versioning.

8.2.6 Header `Prefer`

The `Prefer` header, as defined in **[HTTP-PREFER]**, allows clients to request certain behavior from the service. The service MUST ignore preference values that are either not supported or not know by the service.

The value of the `Prefer` header is a comma separated list of *preferences*. The following subsections describe preferences whose meaning in OData is defined by this specification.

In response to a request containing a `Prefer` header, the service MAY return the `Preference-Applied` Header.

8.2.6.1 Preference `odata.allow-entityreferences`

Clients MAY specify a preference of `odata.allow-entityreferences` in a request to retrieve data from the service. If specified, the service MAY return entity references in place of entities that have previously been returned, with at least the properties requested, in the same response (for example, when serializing the expanded results of many-to-many relationships). The service MUST NOT return entity references in place of requested entities if the client has not specified `odata.allow-entityreferences`.

In the case that the service honors the `odata.allow-entityreferences` preference it MUST include a `Preference-Applied` response header containing the `odata.allow-entityreferences` preference to indicate that entity references MAY be returned in place of entities that have previously been returned.

8.2.6.2 Preference `odata.continue-on-error`

Clients MAY specify a preference of `odata.continue-on-error` on a batch request in order to request that, upon encountering a request within the batch that returns an error, the service return the error for that request and continue processing additional requests within the batch.

If not specified, upon encountering an error the service MUST return the error and stop processing additional requests within the batch.

8.2.6.3 Preference `odata.include-annotations`

Clients MAY specify a preference of `odata.include-annotations` in a request for data or metadata in order to specify a set of desired annotations to be returned in the response.

The value of the `odata.include-annotations` preference is a comma separated list of namespaces or namespace qualified term names to include or exclude in a data or metadata response, and "*" representing all.

The syntax of the `odata.include-annotations` preference is as follows:

```
annotationsPreference = 'odata.include-annotations' EQ DQUOTE annotationsList
DQUOTE
annotationsList = annotationIdentifier *(COMMA annotationIdentifier)
annotationIdentifier = [ excludeOperator]
                       ( STAR
                       / annotationNamespace ["." termName] )
annotationNamespace = namespace ["." STAR]
excludeOperator = "-"
```

The `odata.include-annotations` preference is only a hint to the service. The service MAY ignore the preference and is free to decide whether or not to return annotations not specified in the `odata.include-annotations` preference.

The most specific identifier always takes precedence. If the same identifier value is requested to both be excluded and included the behavior is undefined; the service MAY return or omit the specified vocabulary but MUST NOT raise an exception.

For example, the following `Prefer` header requests all annotations within a metadata document to be returned:

```
    Prefer: odata.include-annotations="*"
```

The following `Prefer` header requests that no annotations are returned:

```
    Prefer: odata.include-annotations="-*"
```

The following `Prefer` header requests that all annotations defined under the "display" namespace (recursively) be returned:

```
Prefer: odata.include-annotations="display.*"
```

The following `Prefer` header requests that the annotation with the term name `subject` within the `display` namespace be returned if applied:

```
Prefer: odata.include-annotations="display.subject"
```

In the case that the client has specified the `odata.include-annotations` preference in the request, the service MAY include a `Preference-Applied` response header containing the `odata.include-annotations` preference to specify the annotations that MAY have been included in the response. This value MAY differ from the annotations requested in the `Prefer` header of the request.

8.2.6.4 Preference `odata.maxpagesize`

Clients MAY specify a preference of `odata.maxpagesize` to request that each collection of entities within the response contain no more than the number of entities specified as the integer value of this preference.

For example, a request for customers and their orders would result in a response containing one collection with customer entities and for every customer a separate collection with order entities. The client could specify `odata.maxpagesize=50` in order to request that each page of results contain a maximum of 50 customers, each with a maximum of 50 orders.

If a collection within the result contains more than the specified `odata.maxpagesize`, the collection SHOULD be a partial set of the results with a next link to the next page of results. The client MAY specify a different value for this preference with every request following a next link.

In the example given, the result page should include a next link for the customer collection, if there are more than 50 customers, and additional next-links for all returned orders collections with more than 50 entries.

If the client has specified the `odata.maxpagesize` preference in the request, and the service limits the number of entities in collections within the response through server-driven paging, the service MAY include a `Preference-Applied` response header containing the `odata.maxpagesize` preference and the maximum page size applied. This value MAY differ from the value requested by the client.

8.2.6.5 Preference `odata.track-changes`

Clients MAY specify a preference of `odata.track-changes` to request that the service return a delta link that can subsequently be used to obtain the changes (deltas) from this result.

For paged results, the preference MUST be specified on each requested page in order to obtain a delta link on the final page of results. The delta link MUST NOT be returned prior to the final page of results.

In the case that the service honors the `odata.track-changes` preference by including a delta link in the final page of a response it MAY include a `Preference-Applied` response header containing the `odata.track-changes` preference. This may be useful in server-driven paging scenarios.

8.2.6.6 Preference `return=representation` and `return=minimal`

As defined in **[HTTP-PREFER]**, clients MAY specify a preference of `return=representation` or `return=minimal`.

In OData, `return=representation` or `return=minimal` may be used on a `POST`, `PUT`, or `PATCH` Data Modification Request other than to a named stream property, or to an Action Request. Specifying a preference of `return=representation` or `return=minimal` in a `GET` or `DELETE` request, or any request to a named stream property, SHOULD return a `4xx Client Error`.

A preference of `return=representation` or `return=minimal` MAY be specified on an individual

Data Modification Request or Action Request within a batch, subject to the same restrictions, but SHOULD return a `4xx Client Error` if specified on the batch request itself.

A preference of `return=minimal` requests that the service invokes the request but does not return content in the response. The service MAY honor this request by returning `204 No Content`. In the case that the service honors the `return=minimal` preference by returning no content, it MAY include a `Preference-Applied` response header containing the `return=minimal` preference.

A preference of `return=representation` requests that the service invokes the request and returns the modified entity. The service MAY honor this request by returning the successfully modified resource in the body of the response, formatted according to the rules specified for the requested format. In the case that the service honors the `return=representation` preference by returning content it MAY include a `Preference-Applied` response header containing the `return=representation` preference.

8.2.6.7 Preference `respond-async`

The `respond-async` preference, as defined in **[HTTP-PREFER]**, allows clients to request that the service process the request asynchronously.

If the client has specified `respond-async` in the request, the service MAY process the request asynchronously and return a 202 Accepted response.

The `respond-async` preference MAY be used in combination with the `wait` preference.

Example: A service receiving the following header might choose to respond

- asynchronously if the synchronous processing of the request will take longer than 10 seconds
- synchronously after 5 seconds
- asynchronously (ignoring the `wait` preference)
- synchronously after 15 seconds (ignoring `respond-async` preference and the `wait` preference)

```
Prefer: respond-async, wait=10
```

8.2.6.8 Preference `wait`

The `wait` preference, as defined in **[HTTP-PREFER]**, can be used to establish an upper bound on the length of time, in seconds, the client is prepared to wait for the service to process the request synchronously once it has been received.

If the `respond-async` preference is also specified, the client requests that the service respond asynchronously after the specified length of time.

If the `respond-async` preference has not been specified, the service MAY interpret the `wait` as a request to timeout after the specified period of time.

8.3 Response Headers

In addition to the Common Headers, a service MAY specify the following response headers.

8.3.1 Header `ETag`

A request that returns an individual resource MAY include an `ETag` header. Services MUST include this header in such a response if they require it to be specified when modifying the resource.

The value specified in the `ETag` header may be specified in the `If-Match` or `If-None-Match` header of a subsequent Data Modification Request or Action Request in order to apply optimistic concurrency in updating, deleting, or invoking the action bound to, the entity.

The value specified in the ETag header may be specified in the `If-Match` or `If-None-Match` header of a subsequent Data Modification Request or Action Request in order to apply optimistic concurrency in updating, deleting, or invoking the action bound to, the entity.

An ETag header MAY also be returned on a metadata document request or service document request to allow the client subsequently to make a conditional request for the metadata or service document. Clients can also compare the value of the ETag header returned from a metadata document request to the metadata ETag returned in a response in order to verify the version of the metadata used to generate that response.

8.3.2 Header `Location`

The Location header MUST be returned in the response from a Create Entity request, to specify the edit URL of the created entity, and in responses returning `202 Accepted` to specify the URL that the client can use to request the status of an asynchronous request.

8.3.3 Header `OData-EntityId`

A response to a create operation that returns `204 No Content` MUST include an `OData-EntityId` response header. The value of the header is the entity-id of the entity that was acted on by the request.

8.3.4 Header `Preference-Applied`

In a response to a request that specifies a `Prefer` header, a service MAY include a `Preference-Applied` header specifying how individual preferences within the request were handled.

The value of the `Preference-Applied` header is a comma-separated list of preferences applied in the response. For more information on the individual preferences, see the `Prefer` header.

8.3.5 Header `Retry-After`

A service MUST include a `Retry-After` header in a `202 Accepted` response; it MAY include it in a `3xx Redirect` response.

The `Retry-After` header specifies the suggested duration of time, in seconds, after which the client SHOULD use the URL returned as the value of the Location header to check on the status of the operation.

9 Common Response Status Codes

An OData service MAY respond to any request using any valid HTTP status code appropriate for the request. A service SHOULD be as specific as possible in its choice of HTTP status codes.

The following represent the most common success response codes. In some cases, a service MAY respond with a more specific success code.

9.1 Success Responses

The following response codes represent successful requests.

9.1.1 Response Code `200 OK`

A `GET`, `PUT`, or `PATCH` request MAY return `200 OK` if the operation is completed successfully. In this case, the response body MUST contain the value of the resource specified in the request URL.

9.1.2 Response Code `201 Created`

A Create Entity, Create Media Entity, Create Link or Invoke Action request that successfully creates a resource MAY return `201 Created`. In this case, the response body MUST contain the resource created.

9.1.3 Response Code `202 Accepted`

A service MAY reply to a Data Service Request with `202 Accepted`, if the client has specified the `respond-async` prefer header. A service MUST NOT reply to a Data Service Request with `202 Accepted` if the request has not included the `respond-async` preference.
`202 Accepted` indicates that the service utilizes an asynchronous processing model; the request has been accepted but has not yet completed. In this case, the response MUST contain a Location header pointing to a status monitor resource in addition to a Retry-After header, and the response body MUST be empty. The Location header URL points to a resource which represents the current state of the asynchronous processing, in the following called the AP resource. As long as the asynchronous processing is not finished, a `GET` request to the AP resource leads to a `202 Accepted` response including a Location header and a Retry-After header (both can differ from the previous response). Always the latest AP resource MUST be used for a further request.

Once the asynchronous processing has successfully completed, the AP resource MUST return `200 OK` with a `Content-Type` header with value `application/http` and a `Content-Transfer-Encoding` header with value `binary` as described in **[HTTP-Message]**. The response body MUST enclose a single HTTP response which is the response to the initial Data Service Request.

A `DELETE` request MAY be sent to the AP resource to terminate the asynchronous processing. A successful deletion will lead to a `200 OK` or to a `204 No Content` response. After a successful deletion the service MUST ensure that no observable change has occurred as a result of the canceled request.

If a `GET` or `DELETE` request to the AP resource fails, the service MUST respond with a HTTP `4xx` or `5xx` status code. The service MAY respond with a `410 GONE` if the client waited too long to request the result of the asynchronous processing or if the asynchronous processing was terminated by a system administrator, a system process, or by a successful `DELETE` request to the AP resource. The service MAY also answer with an unspecific `404 Not Found` in these cases.

9.1.4 Response Code `204 No Content`

As allowed in **[HTTP-Semantics]**, a service MAY respond to a Data Modification Request with `204 No Content`. In this case, the response body MUST be empty.

9.1.5 Response Code `3xx Redirect`

As per **[RFC2616]**, a service MAY respond to any request with a response code of `3xx Redirection`. In this case, the response SHOULD include a Location header, as appropriate, with the URL from which the result can be obtained; it MAY include a Retry-After header.

The service MUST ensure that no observable change has occurred to the state of the service as a result of any request that returns a `3xx`.

9.1.6 Response Code `304 Not Modified`

As per **[RFC2616]**, a service MAY respond to a `GET` request containing an `If-Match` or `If-None-Match` header with `304 Not Modified`. In this case the response SHOULD NOT include other headers in order to prevent inconsistencies between cached entity-bodies and updated headers.

9.2 Client Error Responses

Services SHOULD return `4xx Client Error` in response to client errors such as malformed requests, in addition to requests for non-existent resources such as entity collections, entities, or properties.

The service MUST ensure that no observable change has occurred to the state of the service as a result of any request that returns an error status code.

In the case that a response body is defined for the error code, the body of the error is as defined for the appropriate format.

9.2.1 Response Code `404 Not Found`

If the resource specified by the request URL does not exist, the service SHOULD respond with `404 Not Found`. The response body MAY contain an error according to the requested format providing additional information.

9.2.2 Response Code `405 Method Not Allowed`

If the resource specified by the request URL does not support the request method, the service SHOULD respond with `405 Method Not Allowed`. In this case the response MUST include an `Allow` header containing a list of the valid request methods for the requested resource as specified in **[RFC2616]**.

9.3 Server Error Responses

As specified in **[RFC2616]**, services should use error codes in the `5xx` range to indicate service errors.

9.3.1 Response Code `501 Not Implemented`

If the client requests functionality not implemented by the OData Service, the service MUST respond with `501 Not Implemented` and SHOULD return an OData error describing the functionality not implemented.

9.4 In-Stream Errors

In the case that the service encounters an error after sending a success status to the client, the service MUST generate an error within the payload which may leave the response malformed. Clients MUST treat the entire response as being in error.

This specification does not prescribe a particular format for generating errors within a payload.

10 Metadata URL

The *metadata URL* describes the content of the payload.

Request payloads generally do not require metadata URLs as the type of the payload can generally be determined by the request URL.

For details on how the metadata URL is used to describe a payload, see the relevant sections in the particular format.

The following URL templates describe how the metadata URL is constructed for each category of payload. The format for each template is as follows:

Category	`Metadata URL template`
	`Sample resource URL`
	`Metadata URL for sample`
	Comments

In each example:

`{entity-set}` is the name of an entity set or path to a contained navigation property

`{entity-set-or-type-name}` is the name of an entity-set, path to a contained navigation property, or type-name

`{metadata-url}` is the resource path to the $metadata document

`{namespace}` is the name of a namespace

`{property-list}` is a comma separated list of properties on the resource

`{type-cast}` is a segment containing the qualified name of a derived type

The full grammar for the metadata URL is defined in **[OData-ABNF]**.

Metadata URL Formats

Set of Entities	`{metadata-url}#{entity-set}`
	`http://host/service/Customers`
	`http://host/service/$metadata#Customers`
	If the entity set is in an entity container that has the `IsDefaultEntityContainer` attribute set to true, the entity set MAY be unqualified. If the entity set is in an entity container that has does not have an `IsDefaultEntityContainer` attribute with a value of true, the entity set MUST be namespace and container qualified.

Set of Projected Entities	`{metadata-url}#{entity-set}[/{type-cast}]({property-list})`
	`http://host/service/Customers?$select=Address,Orders`
	`http://host/service/$metadata#Customers(Address,Orders)`
	If a feed contains a subset of properties, a parenthetical comma separated list of properties is appended to the `{entity-set}` to specify the set of properties in the response. If all properties are included for a particular entity, a dot (`.`) MAY be used and is equivalent to omitting the `{property-list}`.

Set of Projected Related Entities	`{metadata-url}#{entity-set}[/{type-cast}]({property-list})`
	`http://host/service/Employees/Sales.Manager?$expand=DirectReports($select=FirstName,LastName;$levels=4)`
=4	`http://host/service/$metadata#Employees/Sales.Manager(.,DirectReports+(FirstName,LastName))`
	If a nested collection contains a subset of properties, a parenthetical comma separated list of properties is appended to the collection property name to specify the set of properties in elements of the nested collection. Additionally, if the pattern is recursive for nested children the `{property-name}` includes the `+` suffix.
Set of Derived Entities	`{metadata-url}#{entity-set}/{type-cast}`
	`http://host/service/Customers/Model.VipCustomer`
	`http://host/service/$metadata#Customers/Model.VipCustomer`
	If a feed consists exclusively of derived elements, a type cast segment is added to the metadata URL.
Element	`{metadata-url}#{entity-set}/@Element`
	`http://host/service/Customers(1)`
	`http://host/service/$metadata#Customers/@Element`
	If a response is guaranteed to contain only a single entry from a feed, `/@Element` is appended to the metadata URL. This is also true for links payloads.
Projected Element	`{metadata-url}#{entity-set}{property-list}/@Element`
	`http://host/service/Customers(1)?$select=Name,Rating`
	`http://host/service/$metadata#Customers(Name,Rating)/@Element`
	If a single entry contains a subset of properties, a parenthetical comma separated list of properties is appended to the metadata URL prior to appending `/@Element` to specify the set of properties in the response.
Derived Element	`{metadata-url}#{entity-set}/{type-cast}/@Element`
	`http://host/service/Customers(2)/Model.VipCustomer`
	`http://host/service/$metadata#Customers/Model.VipCustomer/@Element`
	If a response is guaranteed to contain only a single element of a derived type, `/<Type>/@Element` is appended to the metadata URL.
Complex	`{metadata-url}#{namespace}.{type-name}`

or Primitive Type Property	`http://host/service/Customers(1)/Name`
	`http://host/service/$metadata#Edm.String`
	If a response is a complex type or primitive type, the metadata URL contains the fully qualified type of the property.
Collection Type Property	`{metadata-url}#Collection({namespace}.{type-name})`
	`http://host/service/Customers(1)/Hobbies`
	`http://host/service/$metadata#Collection(Edm.String)`
	If a response is a collection of complex types or primitive types, the metadata URL names the type of the property.
Null	`{metadata-url}#Edm.Null`
	`http://host/service/Customers(1)/Description`
	`http://host/service/$metadata#Edm.Null`
	If a request or response is a null collection, complex or primitive type property, the metadata URL fragment is simply `Edm.Null`.
Operation Result	`{metadata-url}#{entity-set-or-type-name}`
	`http://host/service/TopFiveCustomers`
	`http://host/service/$metadata#Customers`
	A response from an operation (service operation, action, function) has a metadata URL that identifies the type returned by the operation. The metadata URL will correspond to one of the former examples.
Service Doc	`{metadata-url}`
	`http://host/service/`
	`http://host/service/$metadata`
	The metadata URL of the service document is the metadata URL of the service.
Entity Reference	`{metadata-url}#{entity-set}/$ref[/@Element]`
	`http://host/service/Customers(1)/Orders(1)/$ref`
	`http://host/service/$metadata#Orders/$ref/@Element`
	If a response is guaranteed to contain only a single element from a navigation property with cardinality of many, `/@Element` is appended to the metadata URL.
Delta	`{metadata-url}#{entity-set}/@Delta`

Response	`http://host/service/Customers?$deltaToken=1234`
	`http://host/service/$metadata#Customers/@Delta`
	The metadata URL is the same as the metadata URL to the root entity set, followed by `/@Delta`.

11 Data Service Requests

OData Services MAY support the following types of requests.

11.1 Metadata Requests

An OData service is a self-describing service that exposes metadata defining the entity sets, relationships, entity types, and operations.

11.1.1 Service Document Request

Service Documents enable simple hypermedia-driven clients to enumerate and explore the resources offered by the data service.

OData services MUST support returning a service document from the root URL of the service (the *Service Root*).

The format of the Service Document is dependent upon the format selected. For example, in Atom the Service Document is an AtomPub Service Document (as specified in **[RFC5023]**,).

11.1.2 Metadata Document Request

An OData *Metadata Document* is a representation of the data model that describes the data and operations exposed by an OData service.

[OData-CSDL] describes an XML representation for OData Metadata Documents and provides an XML Schema to validate their contents. The media type of the XML representation of an OData Metadata Document is `application/xml`.

OData services MUST expose a Metadata Document that describes the data model exposed by the service. The *Metadata Document URL* SHOULD be the root URL of the service with `$metadata` appended. To retrieve this document the client issues a `GET` request to the Metadata Document URL.

If a request for metadata does not specify a format preference (via Accept header or `$format`) then the XML representation MUST be returned.

11.1.3 Metadata Service Document Request

An OData *Metadata Service* is a representation of the data model that describes the data and operations exposed by an OData service as an OData service with a fixed (meta) data model.

OData services MAY/SHOULD/MUST expose their data model as an *OData Metadata Service*.

The metadata service MUST use the schema defined in **[OData-CSDL]**. The *Metadata Service Document URL* MUST be the metadata document URL of the service with a forward slash appended. To retrieve this document the client issues a `GET` request to the Metadata Service Document URL

11.2 Requesting Data

OData services support requests for data through the use of HTTP `GET` requests.

The path of the URL specifies the target of the request (for example; the collection of entities, entity, navigation property, scalar property, or operation). Additional query operators, such as filter, sort, page, and projection operations are specified through query options.

The format of the returned data is dependent upon the request and the format specified by the client, either in the Accept header or using the `$format` query option.

This section describes the types of data requests defined by OData. For complete details on the syntax for building requests, see **[OData-URL]**.

Clients MUST be prepared to receive additional properties in an entity or complex type instance that are not advertised in metadata, even for types not marked as open.

11.2.1 Evaluating System Query Options

OData defines a number of system query options that may be used to further refine the request. The result of the request MUST be as if the System query options were evaluated in the following order.

Prior to applying any server-driven paging:

- `$search`
- `$filter`
- `$count`
- `$orderby`
- `$skip`
- `$top`

After applying any server-driven paging:

- `$expand`
- `$select`
- `$format`

11.2.2 Requesting Individual Entities

To retrieve an individual entity, the client makes a `GET` request to an *entity request URL*.

The entity request URL MAY be returned in a response payload containing that instance (for example, as a self-link in an **[OData-Atom]** payload).

Services MAY support conventions for constructing an entity request URL using the entity's Key Value(s), as described in **[OData-URL]**.

11.2.3 Requesting Individual Properties

A service SHOULD support retrieving an individual property value.

To retrieve an individual property, the client issues a `GET` request to the property URL. The property URL is the entity request URL with "/" and the property name appended.

For complex typed properties, the path MAY be further extended with the name of the individual property of the complex type.

See **[OData-URL]** for details.

For example:

```
http://services.odata.org/OData/OData.svc/Products(1)/Name
```

11.2.3.1 Requesting a Property's Raw Value using `$value`

A service SHOULD support retrieving the raw value of a primitive type property. To retrieve this value, the client sends a `GET` request to the property value URL. See the **[OData-URL]** document for details.

For example:

```
http://services.odata.org/OData/OData.svc/Products(1)/Name/$value
```

The raw value of an `Edm.Binary` property MUST be serialized as an unencoded byte stream.

The raw value of other properties SHOULD be represented using the `text/plain` media type. See **[OData-ABNF]** for details.

A `$value` request for a property that is `NULL` SHOULD result in a `404 Not Found` response.

11.2.4 Querying Collections

OData services support querying collections of entities.

The target collection is specified through a URL, and query operations such as filter, sort, paging, and projection are specified as *System Query Options* provided as query options. The names of all System Query Options are prefixed with a dollar (`$`) 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 fail any request that contains the unsupported option and SHOULD return `501 Not Implemented`.

11.2.4.1 The `$filter` System Query Option

The set of entities returned MAY be restricted through the use of the `$filter` System Query Option.

For example:

```
http://services.odata.org/OData/OData.svc/Products?$filter=Price lt 10.00
```

Returns all Products whose `Price` is less than $10.00.

$count may be used within a $filter expression to limit the entities returned based on the count of related entities or elements within a collection-valued property:

```
http://services.odata.org/OData/OData.svc/Cateoriges?$filter=Products/$count lt 10
```

Returns all Categories with less than 10 products.

The value of the `$filter` option is a Boolean expression as defined in **[OData-ABNF]**.

11.2.4.1.1 Built-in Filter Operations

OData supports a set of built-in filter operations, as described in this section. For a full description of the syntax used when building requests, see **[OData-URL]**.

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

11.2.4.1.2 Built-in Query Functions

OData supports a set of built-in functions that can be used within `$filter` operations. The following table lists the available functions. For a full description of the syntax used when building requests, see **[OData-URL]**.

OData does not define an ISNULL or COALESCE operator. Instead, there is a `null` literal that can be used in comparisons.

Function	Example
String Functions	
bool substringof	substringof('Alfreds',CompanyName)
bool endswith	endswith(CompanyName,'Futterkiste')
bool startswith	startswith(CompanyName,'Alfr')
int length	length(CompanyName) eq 19
int indexof	indexof(CompanyName,'lfreds') eq 1
string substring	substring(CompanyName,1) eq 'lfreds Futterkiste'
string tolower	tolower(CompanyName) eq 'alfreds futterkiste'
string toupper	toupper(CompanyName) eq 'ALFREDS FUTTERKISTE'
string trim	trim(CompanyName) eq 'Alfreds Futterkiste'
string concat	concat(concat(City,', '), Country) eq 'Berlin, Germany'
Date Functions	
DateTimeOffset now	StartTime ge now()
int day	day(StartTime) eq 8
int hour	hour(StartTime) eq 1
int minute	minute(StartTime) eq 0
int month	month(BirthDate) eq 12
byte second	second(StartTime) eq 0
decimal fractionalseconds	second(StartTime) eq 0
int year	year(BirthDate) eq 0
Date date	date(StartTime) ne date(EndTime)
TimeOfDay time	time(StartTime) le StartOfDay
int totaloffsetminutes	totaloffsetminutes(StartTime) eq 60
DateTimeOffset mindatetime	StartTime eq mindatetime()
DateTimeOffset maxdatetime	EndTime eq maxdatetime()
Math Functions	
double round	round(Freight) eq 32
decimal floor	floor(Freight) eq 32
double ceiling	ceiling(Freight) eq 33
Type Functions	

any cast	cast(ShipCountry,Edm.String)
bool isof	isof(NorthwindModel.Order)
bool isof	isof(ShipCountry,Edm.String)
Geo Functions	
double geo.distance	geo.distance(CurrentPosition, TargetPosition)
double geo.length	geo.length(DirectRoute)
bool geo.intersects	geo.intersects(Position, TargetArea)

11.2.4.1.3 Parameter Aliases

Literal values used in function parameters or within a `$filter` or `$orderby` expression MAY be substituted by a parameter alias. Parameters aliases are names beginning with an at sign (`@`).

Actual parameter values are specified as query options in the query part of the request URL. The query option name is the name of the parameter alias, and the query option value is the value to be used for the specified parameter alias.

For example:

```
GET http://host/service.svc/Employees?$filter=Region eq @p1&@p1='WA'
```

Returns all employees whose Region property matches the string parameter value "WA".

Parameter aliases allow the same value to be used multiple times in a request and may be used to reference primitive values, complex, or collection values.

If a parameter alias is not given a value in the Query part of the request URL, the value MUST be assumed to be null.

11.2.4.2 The `$orderby` System Query Option

The `$orderby` System Query option specifies the order in which entities are returned from the service.

The value of the `$orderby` System Query option contains a comma separated list of expressions whose primitive result values are used to sort the results. A special case of such an expression is a property path terminating on a primitive property. A type cast using the qualified entity type name is required to order by a property defined on a derived type.

The expression MAY include the suffix `asc` for ascending or `desc` for descending, separated from the property name by one or more spaces. If `asc` or `desc` is not specified, the service MUST order by the specified property in ascending order.

Null values come before non-null values when sorting in ascending order and after non-null values when sorting in descending order.

For example:

```
http://services.odata.org/OData/OData.svc/Products?$orderby=ReleaseDate asc,
Rating desc
```

Returns all Products ordered by release date in ascending order, then by rating in descending order.

Related entities may be ordered by specifying $orderby within the $expand clause:

```
http://services.odata.org/OData/OData.svc/Categories?$expand=Products($orderby
=ReleaseDate asc, Rating desc)
```

Returns all Categories, and their Products ordered according to release date and in descending order of rating.

`$count` may be used within a `$orderby` expression to order the returned entities according to the count of related entities or elements within a collection-valued property:

```
http://services.odata.org/OData/OData.svc/Cateoriges?$orderby=Products/$count
```

Returns all Categories ordered by the number of Products within each category.

11.2.4.3 The $top System Query Option

The $top System Query Option specifies a non-negative integer that limits the number of entities returned within a collection. The service MUST return the number of available entities up to, but not exceeding, the specified value.

For example:

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

Would return only the first five Products in the Products entity set.

If no unique ordering is imposed through an $orderby query option, the service MUST impose a stable ordering across requests that include $top.

11.2.4.4 The $skip System Query Option

The $skip System Query Option specifies that the result MUST NOT include the first n entities, where n is a non-negative integer value specified by the $skip query option.

For example:

```
http://services.odata.org/OData/OData.svc/Products?$skip=5
```

Would return Products starting with the 6th Product in the Products entity set.

Where $top and $skip are used together, the $skip MUST be applied before the $top, regardless of the order in which they appear in the request.

For example:

```
http://services.odata.org/OData/OData.svc/Products?$top=5&$skip=2
```

Would return the third through seventh Products in the Products entity set.

If no unique ordering is imposed through an $orderby query option, the service MUST impose a stable ordering across requests that include $skip.

11.2.4.5 The $count System Query Option

The $count System Query Option with a value of true specifies that the total count of entities within a collection matching the request MUST be returned along with the result.

For example:

```
http://services.odata.org/OData/OData.svc/Products?$count=true
```

Would return, along with the results, the total number of products in the set.

The count of related entities can be requested by specifying the $count query option within the $expand clause:

```
http://services.odata.org/OData/OData.svc/Categories?
    $expand=Products($count=true)
```

An $count query option with a value of false (or not specified) hints that the service SHOULD NOT return a count.

The service MUST return an HTTP Status code of 404 Bad Request if a value other than true or false is specified.

The $count system query option ignores any $top, $skip, or $expand query options, and returns the total count of results across all pages including only those results matching any specified $filter.

How the count is encoded in the response body is dependent upon the selected format.

11.2.4.6 The $search System Query Option

The $search System Query Option MAY be used to restrict the result to include only those entities matching the specified search expression.

For example:

```
http://services.odata.org/OData/OData.svc/Products?$search=bike
```

Returns all Products that match the search term "bike". The definition of what it means to match is dependent upon the implementation.

The search expression MAY contain phrases, enclosed in double-quotes.

```
http://services.odata.org/OData/OData.svc/Products?$search="mountain bike"
```

Returns all Products that match the phrase "mountain bike".

The search expression MAY contain the upper case keyword NOT in order return entities that do not match the specified term.

```
http://services.odata.org/OData/OData.svc/Products?$search=NOT clothing
```

Returns all Products that do not match "clothing".

The search expression MAY contain multiple terms, separated by a space (implicit AND) or the upper-case keyword AND, indicating that all such terms must be matched:

```
http://services.odata.org/OData/OData.svc/Products?$search=mountain AND bike
```

Returns all Products that match both "mountain" and "bike".

The search expression MAY contain the upper-case keyword OR in order return entities that satisfy any of the expressions

```
http://services.odata.org/OData/OData.svc/Products?$search=mountain OR bike
```

Returns all Products that match either "mountain" or "bike".

The search expression MAY contain parenthesis to group together multiple expressions.

```
http://services.odata.org/OData/OData.svc/Products?$search=(mountain OR bike)
AND NOT clothing
```

Returns all Products that match either "mountain" or "bike" and do not match clothing.

The operations within a search expression MUST be evaluated in the following order: grouping operator, NOT operator, AND operator, OR operator

If both $search and $filter are specified in the same request, only those entities satisfying both criteria are returned.

The value of the $search option is a Boolean expression as defined in **[OData-ABNF]**.

11.2.4.7 Server-Driven Paging

OData services MAY return only a partial set of the entities identified by the request URL. In that case the response MUST contain a link that allows retrieving the next partial set of entities. This link is called a next-link; its representation is format-specific. The final partial set of entities MUST NOT contain a next-link.

OData clients MUST treat the URL of the next-link as opaque, and especially MUST NOT apply/append system query options to the URL of a next-link, as these system query options MAY already have been used by the service when building the next-link.

The client MAY request a maximum page size through the `odata.maxpagesize` preference. The service MAY honor this requested page size or MAY implement a page size different than, or in the absence of, this preference.

OData services MAY use the reserved system query option `$skiptoken` when building next-links. Its content is opaque, service-specific, and must only follow the rules for URL query parts.

OData clients MUST NOT use the system query option `$skiptoken` when constructing requests.

11.2.5 Specifying Properties to Return

The `$select` and `$expand` system query options enable the client to specify the set of properties and navigation property included in a response.

11.2.5.1 The `$expand` System Query Option

The `$expand` system query option indicates the related entities that MUST be represented inline.

The value of the `$expand` query option MUST be a comma separated list of navigation property names, optionally followed by a parenthetical set of expand options for filtering, sorting, selecting, paging, or expanding the related entities.

For a full description of the syntax used when building requests, see **[OData-URL]**.

For a full description of the syntax used when building requests, see **[OData-URL]**.

Examples:

```
http://host/service.svc/Customers?$expand=Orders
```

For each customer entity within the Customers entity set, the value of all associated Orders MUST be represented inline.

```
http://host/service.svc/Customers?$expand=Orders/$ref
```

Entity references to the associated Orders are inlined instead of the associated Orders themselves.

```
http://host/service.svc/Customers?$expand=Orders($filter=Amount gt 100)
```

For each customer entity within the Customers entity set, the value of those associated Orders whose Amount is greater than 100 MUST be represented inline.

```
http://host/service.svc/Orders?$expand=OrderLines($expand=Product),Customer
```

For each Order within the Orders entity set, the following MUST be represented inline:

- The Order Lines associated with the Orders identified by the resource path section of the URL and the products associated to each Order line.
- The Customer associated with each Order returned.

```
http://host/service.svc/Customers?$expand=SampleModel.VipCustomer/InHouseStaff
```

For each Customer entity in the Customers entity set, the value of all associated `InHouseStaff` MUST be represented inline if the entity is of type `VipCustomer` or a subtype of that. For entities that are not of type `VipCustomer`, or any of its subtypes, that entity SHOULD be returned with no inline representation for the expanded navigation property.

11.2.5.1.1 The `$levels` Expand Option

The $levels expand option can be used within $expand to specify the number of levels of recursion for a hierarchy in which the related entity type is the same as, or can be cast to, the source entity type. The same expand options are applied at each level of the hierarchy.

```
http://contoso.com/HR/Employees?$expand=Model.Manager/DirectReports($levels=4)
```

Return each employee from the Employees entity set and, for each employee that is a manager, return all direct reports, recursively to four levels.

11.2.5.2 The `$select` System Query Option

The `$select` system query option requests that the service return only the properties, dynamic properties, actions and functions explicitly requested by the client. The service MUST return the specified content, and MAY choose to return additional information.

The value of the `$select` query option is a comma separated list of properties, qualified action names, qualified function names, the star operator (`*`), or the star operator prefixed with the name of the entity container in order to specify all operations within the container.

For example, the following request returns just the Rating and ReleaseDate for the matching Products:

```
http://services.odata.org/OData/OData.svc/Products?$select=Rating,ReleaseDate
```

It is also possible to request all properties, including any dynamic properties, using a star request:

```
http://services.odata.org/OData/OData.svc/Products?$select=*
```

A star request SHOULD NOT introduce navigation properties, actions or functions not otherwise requested.

Properties of related entities MAY be specified by including the $select query option within the $expand:

```
http://services.odata.org/OData/OData.svc/Products?
$expand=Category($select=Name)
```

If a $select is present, it MUST include any expanded navigation properties in order for the related entities to be included in the result.

```
http://services.odata.org/OData/OData.svc/Categories?
$select=CategoryName,Products&$expand=Products
```

It is also possible to request all actions or functions available for each returned entity:

```
http://services.odata.org/OData/OData.svc/Products?$select=DemoService.*
```

11.2.6 The `$format` System Query Option

A request with a `$format` system query option specifies that the response MUST use the media type specified by the query option.

If the `$format` query option is present in a request, it MUST take precedence over the value(s) specified in the `Accept` request header.

The value of the `$format` query option MAY be any valid internet media type, which MAY include parameters.

For example:

```
http://services.odata.org/OData/OData.svc /Orders?
                            $format=application/json;odata.metadata=full
```

is equivalent to a request with an `Accept` header using the same media type; it requests the set of Order entities represented using the JSON media type including full metadata, as specified in **[OData-JSON]**.

In addition, format-specific abbreviations may be used, see **[OData-Atom]** and **[OData-JSON]**.

Media type parameters MUST NOT be appended to the format abbreviations.

For example:

```
http://services.odata.org/OData/OData.svc/Orders?$format=json
```

is equivalent to a request with the `Accept` header set to `application/json`; it requests the set of Order entities represented using the JSON media type with minimal metadata, as specified in **[OData-JSON]**.

11.2.7 Requesting Related Entities

To request related entities according to a particular relationship, the client issues a `GET` request.

The path of the request is the source entity's request URL, followed by a forward slash and the name of the navigation property representing the relationship.

If the navigation property does not exist on the entity indicated by the request URL, the service MUST return `404 Not Found`.

If the relationship terminates on a collection, the response MUST be the format-specific representation of the collection of related entities. If no entities are related, the response is the format-specific representation of an empty collection.

If the relationship terminates on a single entity, the response MUST be the format-specific representation of the related single entity. If no entity is related, the response MUST be `204 No Content`.

For example:

```
http://services.odata.org/OData/OData.svc/Products(1)/Supplier
```

Returns the supplier of the product with `ID=1` in the Products entity set.

11.2.8 Requesting Entity References

To request entity references in place of the actual entities, the client issues a `GET` request with `/$ref` appended to the resource path.

If the resource path does not resolve to an entity or collection of entities, the service MUST return `404 Not Found`.

If the resource path terminates on a collection, the response MUST be the format-specific representation of a collection of entity references pointing to the related entities. If no entities are related, the response is the format-specific representation of an empty collection.

If the resource path terminates on a single entity, the response MUST be the format-specific representation of an entity reference pointing to the related single entity. If no entity is related, the response MUST be `204 No Content`.

For example:

```
http://services.odata.org/OData/OData.svc/Products(0)/Orders/$ref
```

Returns a collection containing an entity reference for each Order related to the Product with `ID=0`.

11.2.9 Requesting the Number of Elements of a Collection

To request only the number of elements of a collection of entities or elements of a collection-valued property, the client issues a `GET` request with `/$count` appended to the path of the request URL.

On success, the response body MUST contain the count of entities matching the request after applying any `$filter` or `$search` system query options, formatted as a simple scalar integer value. The returned count MUST NOT be affected by `$top`, `$skip`, `$orderby`, or `$expand`.

For example:

```
http://services.odata.org/OData/OData.svc/Products/$count
```

Returns the count of Products in the Products entity set.

```
http://services.odata.org/OData/OData.svc/Customers('ALFKI')/Interests/$count
```

Returns the count of interests within the Interests collection of customer 'ALFKI'.

11.3 Requesting Changes

An OData service MAY support requesting changes (deltas) from a previous result.

Services MAY support requesting changes for arbitrary queries against the entity set or MAY require that any filters be applied solely to immutable (i.e., key) fields.

Clients request that the service support tracking changes to a result by specifying the `odata.track-changes` preference on a request. In response, the service MAY include a *delta-link* for requesting subsequent changes to the result.

Delta links are opaque, service-generated links that the client can use to retrieve subsequent changes to a result.

Delta links are based on a *defining query* that describes the set of results for which changes are being tracked; for example, the request that generated the results containing the delta link. The delta link MUST conceptually encode the following information.

- The collection of entities for which changes are being tracked, along with a starting point from which to track changes.
    - If the defining query contains a `$filter` or `$search`, the response MUST include only changes to entries matching the specified criteria. Added entries MUST be returned for entities that were added or changed and now match the specified criteria, and deleted entries MUST be returned for entities that are changed to no longer match the criteria. Clients MUST NOT append `$filter` or `$search` to a delta link.
    - If the defining query includes expanded relationships, the delta link MUST return changes, additions, or deletions to the expanded entities, as well as added or deleted relationships to expanded entities. Clients MUST NOT append `$expand` to a delta link.
- Entities are considered changed if any of the structural properties have changed. Changes to related entities and to streams are not considered a change to the entity containing the stream or navigation property. If the defining query contains a projection, the generated delta link SHOULD logically include the same projection, such that the delta query SHOULD only return fields specified in the projection. However, services MUST NOT use the projection in determining which entries have changed; the entry MUST be returned whether or not the field that changed was specified in the projection. Client MUST NOT append `$select` to a delta link.

The delta link MUST NOT encode the following query options:

- Delta links represent changes to an entire conceptual set of results, not just an individual page, and MUST NOT encode any client top or skip value. The client may append `$top` and `$skip` to the delta link in order to page through the results of a query. In this case, the delta link for the next set of changes appears on the last page of results. There may be no changes in the final page if the specified value for `$skip` is equal to or greater than the number of changes available.
- Delta links SHOULD NOT encode a request for an inline count. The client MAY append the `$count` system query option to the delta link in order to retrieve the number of added, changed, or deleted entities represented by the delta link. The count MUST NOT include added or deleted links.
- Delta links MUST NOT encode a response format. The client MAY append `$format` to the delta link in order to request a particular format for the response.
- The results of a delta query are ordered by the service in such a way as to guarantee consistency when applied in the order provided. The client MUST NOT append the `$orderby` query option to the delta link.

Clients MAY append a `/$count` segment to the path of a delta link in order to get just the number of changes available. The count MUST include all added, changed, or deleted entries and MUST NOT include added or deleted links.

The results of a delta link request MAY be returned in more than one page by the service. If more than one page of changes is available, each page MUST contain a link for the next page of changes.

The final page of changes SHOULD contain a new delta link for subsequent changes. This delta link MUST return all changes subsequent to the last change of the previous delta link and MUST NOT be returned prior to the final page of results.

Services SHOULD return only changed entries, but MAY return additional entities matching the defining query for which the client will not see a change.

If the delta link is no longer valid, the service SHOULD respond with `410 Gone`, and SHOULD include the URL for refetching the entire set in the `location` header of the response.

11.4 Data Modification

An OData service MAY support Create, Update, and Delete operations for some or all of the entities that it exposes. Additionally, services MAY support one or more Actions which may affect the state of the system.

A successfully completed Data Modification Request must not violate the integrity of the data.

The client may request whether content be returned from a Create, Update, or Delete request, or the invocation of an Action, by specifying the Prefer header.

11.4.1 Common Data Modification Semantics

Data Modification Requests share the following semantics.

11.4.1.1 Use of ETags for Avoiding Update Conflicts

The client MAY include an ETag value in an `if-match` or `if-none-match` request header of a Data Modification Request or Action Request. If specified, the operation MUST only be invoked if the `if-match` or `if-none-match` condition is satisfied.

The ETag value specified in the `if-match` or `if-none-match` request header may be obtained from an ETag header of a request for an individual entity, or may be included for an individual entry in a format-specific manner.

11.4.1.2 Handling of DateTimeOffset Values

Services SHOULD preserve the offset of `Edm.DateTimeOffset` values, if at all possible. However, where the underlying storage does not support offset services MAY be forced to normalize the value to some common time zone (i.e. UTC) in which case the result would be returned with that time zone offset.

11.4.1.3 Handling of Properties Not Advertised in Metadata

Clients MUST be prepared to receive additional properties in an entity or complex type instance that are not advertised in metadata, even for types not marked as open. By using `PATCH` when updating entities, clients can ensure that such properties values are not lost if omitted from the update request.

11.4.1.4 Handling of Consistency Constraints

Services MAY impose cross-entity consistency constraints. Certain referential constraints, such as requiring an entity to be created with related entities MAY be satisfied through creating or linking related entities when creating the entity. Other constraints MAY require multiple changes to be specified together in a single atomic change set.

11.4.2 Create an Entity

To create an entity in a collection, the client sends a `POST` request to that collection's URL. The `POST` body MUST contain a single valid entity representation.

An entity may also be created as the result of an Upsert operation.

If the target URL for the collection is a navigation link the corresponding navigation property MUST contain the related entities (`ContainsTarget="true"`) or have an associated navigation property binding to specify the target set of the related entity(ies).

To create an *open entity* (an instance of an open type), additional property values beyond those specified in the metadata MAY be sent in the request body. The service MUST treat these as dynamic properties and add them to the created instance.

If the entity being created is not an open entity, additional property values beyond those specified in the metadata SHOULD NOT be sent in the request body. The service MUST fail if unable to persist all property values specified in the request.

Upon successful completion, the response MUST contain a Location header that contains the edit URL of the created entity.

Upon successful completion the service MUST respond with either `201 Created`, or `204 No Content` if the request included a Prefer header with a value of `return=minimal`.

11.4.2.1 Link to Related Entities When Creating an Entity

A service SHOULD support linking new entities to existing entities upon creation.

To create a new entity with links to existing entities in a single request, the client includes the entity-ids of the related entities associated with the corresponding navigation properties in the request body.

The representation for binding information is format-specific.

For example, in JSON the client can create a new manager entity with links to two existing employees by applying the odata.bind annotation to the DirectReports navigation property:

```
{
  "odata.type":"Northwind.Manager",
  "EmployeeID": 1,
  "DirectReports@odata.bind": [
    "http://services.odata.org/OData/OData.svc/Employees(5)"
    "http://services.odata.org/OData/OData.svc/Employees(6)"
}
```

In Atom the client can create a new manager entity with links to existing to two existing employees by including a navigation link element for each employee in the Atom entry representing the manager:

```
<entry>
   <id>http://servcies.odata.org/OData/OData.svc/Employees(1)</id>
   <title type="text" />
   <updated>2011-02-16T01:00:25Z</updated>
   <author><name /></author>
   <link
    rel="http://docs.oasis-open.org/odata/ns/related/DirectReports"
    href="http://services.odata.org/OData/OData.svc/Employees(5)"
    type="application/atom+xml;type=entry"
    title="DirectReports"
   />
   <link
    rel="http://docs.oasis-open.org/odata/ns/related/DirectReports"
    href="http://services.odata.org/OData/OData.svc/Employees(6)"
    type="application/atom+xml;type=entry"
    title="DirectReports"
   />
   <category term="NorthwindModel.Manager" scheme="http://odata.org/scheme"/>
   <content type="application/xml">
     <metadata:properties>
       <data:EmployeeID>1</data:EmployeeID>
     </metadata:properties>
   </content>
</entry>
```

Upon successful completion of the operation, the service MUST create the requested entity and relate it to the requested existing entities.

Upon failure of the operation, the service MUST NOT create the new entity. In particular, the service MUST never create an entity in a partially-valid state (with the navigation property unset).

11.4.2.2 Create Related Entities When Creating an Entity

A service that supports creating entities SHOULD support creating related entities as part of the same request.

A request to create an entity MAY specify related entities that should also be created. The related entities MUST be represented using the appropriate inline representation of the navigation property. The navigation property MUST contain the related entities (`ContainsTarget="true"`) or have an associated navigation property binding to specify the target set of the related entity(ies).

Properties computed by the service, along with properties of related entities that are tied to key properties of the principal entity by a referential constraint, MAY be omitted from the request. If the inline representation contains a value for a computed or dependent property, the service MUST ignore that value when creating the related entity.

On success, the service MUST create each entity and relate them.

On failure, the service MUST NOT create any of the entities.

11.4.3 Update an Entity

Services SHOULD support `PATCH` as the preferred means of updating an entity. `PATCH` provides more resiliency between clients and services by directly modifying only those values specified by the client.

The semantics of `PATCH`, as defined in **[RFC5789]**, are to merge the content in the request payload with the [entity's] current state, applying the update only to those components specified in the request body. Collection properties and primitive properties provided in the payload MUST replace the value of the corresponding property in the entity or complex type. Missing properties of the containing entity or complex property, including dynamic properties, MUST NOT be directly altered.

Services MAY additionally support `PUT`, but should be aware of the potential for data-loss in round-tripping properties that the client may not know about in advance, such as open or added properties, or properties not specified in metadata. Services that support `PUT` MUST replace all property values with those specified in the request body. Missing properties MUST be set to their default values. Missing dynamic properties MUST be removed or set to NULL.

Key and other non-updatable properties MAY be omitted from the request. If the request contains a value for a key or other non-updatable property, the service MUST ignore that value when applying the update.

The entity MUST NOT contain related entities as inline content.

If the entity being updated is open, then additional values for properties beyond those specified in the metadata or returned in a previous request MAY be sent in the request body. The service MUST treat these as dynamic properties.

If the entity being updated is not open, then additional values for properties beyond those specified in the metadata or returned in a previous request SHOULD NOT be sent in the request body. The service SHOULD consider such a request malformed.

On success, the response MUST be a valid success response.

11.4.4 Upsert an Entity

An Upsert occurs when the client sends an update request to a valid URL that identifies a single entity that does not exist. In this case the service MUST handle the request as a create entity request or fail the request all together.

Upserts are not supported against media entities. Services MUST fail an update request to a URL that identifies an entity that does not exist.

To ensure that an update request is not treated as an insert, the client MAY specify an `If-Match` header in the update request. The service MUST NOT treat an update request containing an `If-Match` header as an insert.

A `PUT` or `PATCH` request MUST NOT be treated as an update if an `If-None-Match` header is specified with a value of `"*"`.

11.4.5 Delete an Entity

To delete an existing entity, the client sends a `DELETE` request to that entity's edit URL. The request body SHOULD be empty.

On successful completion of the delete, the response MUST be `204 No Content` and contain an empty body.

Services MUST implicitly remove relations to and from an entity when deleting it; clients need not delete the relations explicitly.

Services MAY implicitly delete or modify related entities if required by integrity constraints. If integrity constraints are declared in `$metadata` using a `ReferentialConstraint` element, services MUST modify affected related entities according to the declared integrity constraints, e.g. by deleting dependent entities, or setting dependent properties to `null` or their default value.

11.4.6 Modifying Relationships Between Entities

Relationships between entities are represented by navigation properties as described in Data Model. URL conventions for navigation properties are described in **[OData-URL]**.

11.4.6.1 Create a New Reference to an Existing Entity in a Collection-Valued Navigation Property

To relate an existing entity to another entity, send a `POST` request to the URL for the appropriate navigation property's references collection. The request body MUST contain an entity reference that identifies the entity to be added.

The body MUST be formatted as a single entity reference. See the appropriate format document for details.

On successful completion, the response MUST be `204 No Content` and contain an empty body.

11.4.6.2 Remove a Reference to an Entity

To remove a link to a related entity, the client sends a `DELETE` request to the URL that represents the reference to the related entity.

The `DELETE` request MUST follow not violate any integrity constraints in the data model.

On successful completion, the response MUST be `204 No Content` and contain an empty body.

11.4.6.3 Change the Reference in a Single-Valued Navigation Property

To relate an existing entity to a different existing entity, the client sends a `PUT` request to the URL for the appropriate navigation property's references resource. The request body MUST contain an entity reference that identifies the existing entity to be related, in the same format as for creating a new link between existing entities.

On successful completion, the response MUST be `204 No Content` and contain an empty body.

If a navigation property is nullable, then a change MAY be performed by first removing the existing relationship and then adding the new one. Use the approach described for adding and removing links.

Alternatively, a relationship MAY be updated as part of an update to the source entity by including the required binding information for the new target entity. This binding information MUST be formatted as for a deferred navigation property in a response.

11.4.7 Managing Media Entities

A *media entity* is an entity that represents an out-of-band stream, such as a photograph.

A media entity MUST have a source URL that can be used to read the media stream, and MAY have an `edit-media` URL that can be used to write to the media stream.

Because a media entity has both a media stream and standard entity properties special handling is required.

11.4.7.1 Creating a Media Entity

To create a media entity, the client sends a `POST` request to the media entity's entity set. The request body MUST contain the media value (for example, the photograph) whose media type MUST be specified in a `Content-Type` header.

On successful creation of the media, the service MUST respond with `201 Created` and a response body containing the newly created media entity.

11.4.7.2 Editing a Media Entity Stream

To change the data for a media entity stream, the client sends a `PUT` request to the edit URL of the media entity.

If the entity includes an ETag value for the media stream, the client MUST include an `If-Match` header with the ETag value.

The request body MUST contain the new media value for the entity whose media type MUST be specified in a `Content-Type` header.

If the request to edit a Media Entity Stream returns a non-empty response body, the response body MUST contain the updated media entity.

11.4.7.3 Deleting a Media Entity

To delete a media entity, the client sends a `DELETE` request to the entity's edit URL as described in Delete An Entity.

Deleting a media entity also deletes the media associated with the entity.

11.4.8 Managing Named Stream Properties

An entity may have one or more *named stream properties*. Named stream properties are properties of type `Edm.Stream`.

The values for named stream properties do not appear in the entity payload. Instead, the values are read or written through URLs.

Named streams are not deletable or directly creatable by the client. The service owns their lifetime. The client can request to set the stream data to empty (0 bytes).

11.4.8.1 Editing Named Stream Values

To change the data for a named stream, the client sends a `PUT` request to the edit URL.

If the stream metadata includes an ETag value, the client SHOULD include an `If-Match` header with the ETag value.

The request body MUST contain the new media value for the stream whose media type MUST be specified in a `Content-Type` header.

11.4.9 Managing Values and Properties Directly

Values and properties can be explicitly addressed with URLs. This allows them to be individually modified. See **[OData-URL]** for details on addressing.

11.4.9.1 Update a Primitive Property

To update a value, the client sends a `PUT` request to an edit URL for a primitive property. The message body MUST contain the new value, formatted as a single property according to the specified format.

The same rules apply whether this is a regular property or a dynamic property.

Upon successful completion of the update, the response MUST be a valid update response.

11.4.9.2 Set a Value to Null

There are two ways to set a primitive value to `NULL`. The client can Update a Primitive Property, specifying a null value. Alternatively, the client can send a `DELETE` request with an empty message body to the property URL.

The service SHOULD consider a `DELETE` request to a non-nullable value to be malformed.

The same rules apply whether the target is the value of a regular property or the value of a dynamic property. A missing dynamic property is defined to be the same as a dynamic property with value `NULL`. All dynamic properties are nullable.

On success, the service MUST respond with `204 No Content` and an empty body.

11.4.9.3 Update a Complex Type

To update a complex type, the client SHOULD send a `PATCH` request to that property's URL. The request body MUST contain a single valid representation for the target complex type.

The service MUST directly modify only those properties of the complex type specified in the payload of the `PATCH` request.

The service MAY additionally support clients sending a `PUT` request to a URL that specifies a complex type. In this case, the service MUST replace the entire complex property with the values specified in the request body and set all unspecified properties to their default value.

On success, the response MUST be a valid update response.

11.4.9.4 Update a Collection Property

To update a collection value, the client sends a `PUT` request to the collection property's URL. The message body MUST contain the desired new value, formatted as a collection property according to the specified format.

The service MUST replace the entire value with the value supplied in the request body.

Since collection members have no individual identity, `PATCH` is not supported for collection properties. Services MUST treat collection properties as a unit.

On success, the response MUST be a valid update response.

11.5 Operations

Services MAY support custom operations. Operations (Actions and Functions) are represented as `Action`, `ActionImport`, `Function`, and `FunctionImport` elements in **[OData-CSDL]**.

11.5.1 Common rules for all operations

All operations MUST follow the rules outlined in **[OData-CSDL]**.

11.5.1.1 Declaring Operations in Metadata

Actions and Functions SHOULD be declared in `$metadata`. Actions are declared using an `Action` element while Functions are declared using a `Function` element. Both elements specify the name, parameters, and return types of the operation.

Actions and Functions MAY be bound to an entity container by declaring an `ActionImport` or `FunctionImport` in `$metadata`.

[OData-CSDL] specifies how actions, functions, action imports, and function imports are syntactically defined in CSDL.

For example this `Action` element describes an action that creates an order for a customer using the specified quantity and discount code. This action can be bound to any resource path that represents a `Customer` entity:

```
<Action Name="CreateOrder" IsBindable="true">
    <Parameter Name="customer" Type="SampleModel.Customer">
    <Parameter Name="quantity" Type="Edm.Int32">
    <Parameter Name="discountCode" Type="Edm.String">
</Action>
```

The following `Function` describes a function called `MostRecent` that returns the most recent Order within a collection of Orders:

```
<Function Name="MostRecent" EntitySetPath="orders"
          ReturnType="SampleModel.Order"
          IsBindable="true">
    <Parameter Name="orders" Type="Collection(SampleModel.Order)"/>
</Function>
```

11.5.1.1.1 Binding Operations

Actions and Functions MAY be bound to an entity type, primitive type, complex type, or a collection. The first parameter of a bindable operation is the *binding parameter*.

The namespace (or alias) qualified name of the operation may be appended to any URL that identifies a resource whose type matches, or is derived from, the type of the binding parameter. The resource identified by that URL is used as the *binding parameter value*.

For example, the function

```
<Function Name="MostRecentOrder" ReturnType="SampleModel.Order"
                IsBindable="true" IsAlwaysBindable="true">
    <Parameter Name="customer" Type="SampleModel.Customer" />
</Function>
```

can be bound to any URL that identifies a `SampleModel.Customer`, for example:

```
GET http://server/Customers(6)/SampleModel.MostRecentOrder()
```

invokes the `MostRecentOrder` function with the 'customer' or binding parameter value being the entity identified by `http://server/Customers(6)/`.

11.5.1.1.2 Entity Set Path Expression

For Functions or Actions that return an entity or collection of entities, the entity set associated with the returned entity or collection of entities MAY depend upon the entity set of one of the parameter values used to invoke the operation.

When such a dependency exists an *entity set path expression* is used. An entity set path expression MUST begin with the name of a parameter to the operation, and optionally includes a series of navigation path segments (using type casts, as appropriate) to describe the entity set associated with the returned entity or collection of entities relative to the input parameter.

11.5.1.2 Returning Available Operations within a Payload

Services MAY return the available actions and/or functions bound to a particular entity as part of the entity representation within the payload. The representation of an action or function depends on the format.

For example, given a `GET` request to `http://server/Customers('ALFKI')`, the service might respond with a Customer that includes the `SampleEntities.MostRecentOrder` function bound to the entity:

```
{
  "odata.metadata": ...,
  "#SampleEntities.MostRecent": {
    "title": "Most Recent Order",
    "target": "Customers('ALFKI')/SampleEntities.MostRecentOrder"
  },
  "CustomerID": "ALFKI",
  "CompanyName": "Alfreds Futterkiste",
  ...
}
```

An efficient format that assumes client knowledge of metadata SHOULD NOT include actions and functions in the payload that are available on all instances and whose target URL can be computed via metadata following standard conventions defined in **[ODATA-URL]**.

11.5.2 Functions

Functions are operations exposed by an OData service that MUST return data and MUST have no observable side effects.

11.5.2.1 Invoking a Function

To invoke a function bound to a resource, the client issues a `GET` request to a function URL. A function URL may be obtained from a previously returned entity instance or constructed by appending the namespace (or alias) qualified function name to a URL that identifies a resource whose type is the same as, or derives from, the type of the binding parameter of the function. The value for the binding parameter is the value of the resource identified by the URL prior to appending the function name, and additional parameter values are specified using either inline parameter or parameter name syntax.

Functions may be used within `$filter` or `$orderby` system query options, Such functions may be bound to a resource, as described above, or called directly by specifying the namespace (or alias) qualified function name. Parameter values for functions within `$filter` or `$orderby` MUST be specified according to the inline parameter syntax.

To invoke a function through a function import the client issues a `GET` a request to a URL identifying the function import and passing parameter values using either inline parameter or parameter name syntax. The canonical URL for a function import is the service root, followed by the name of the function import. If the function import is defined in the default entity container, its unqualified name MAY be used, otherwise its namespace- (or alias-) and container-qualified name MUST be used.

Additional path segments and system query options may be appended to URLs that identify composable functions (or function imports) as appropriate for the type returned by the function or function import.

Services MAY support invoking a function or function import using a `PUT`, `POST`, `PATCH` or `DELETE` requests where the target of the operation is the result of the function.

Parameter values passed to functions MUST be specified either as a URL literal (for Primitive Types) or as a JSON formatted OData object (for Complex Types or Collections of Primitive Types or Complex Types).

Function imports MUST NOT be used inside either the `$filter` or `$orderby` system query options.

11.5.2.1.1 Inline Parameter Syntax

Parameter values MAY be specified inline by appending a comma separated list of parameter values, enclosed by parenthesis, i.e. `(` and `)`, to the function name.

Each parameter value MUST be represented as a name/value pair in the format `Name=Value`, where `Name` is the name of the parameter to the function and `Value` is the parameter value.

For example this request:

```
GET http://server/EmployeesByManager(ManagerID=3)
```

Invokes a `Sales.EmployeesByManager` function which takes a single `ManagerID` parameter via the function import `EmployeesByManager`.

And this request:

```
GET http://server/Customers?$filter=Sales.SalesRegion(City=$it/City) eq
"Western"
```

Returns `Customers` whose City property returns "Western" when passed to the `Sales.SalesRegion` function.

Parameter values MAY be specified by substituting a parameter alias in place of an inline parameter to a function call and specifying a value for the alias as a separate query option.

For example:

```
GET http://server/Sales.EmployeesByManager(ManagerID=@p1)?@p1=3
```

Invokes a `Sales.EmployeesByManager` function, passing 3 for the `ManagerID` parameter.

11.5.2.1.2 Parameter Name Syntax

Parameter values for the last Function call in a Request URL Path MAY be specified by appending name/value pairs, representing each parameter Name and Value for that Function, as query strings to the Query part of the Request URL.

This enables clients to invoke functions without parsing the advertised Target Url.

For example:

```
GET http://server/Sales.EmployeesByManager?ManagerID=3
```

11.5.2.2 Function overload resolution

Function overloads are supported in OData, meaning a service MAY expose multiple functions with the same name that take a different set of parameters. The combination of the function name and the unordered list of parameter types and names MUST identify a particular function overload.

Clients SHOULD use typed null values (e.g. null'Date') when passing null parameter values in order to disambiguate requests.

If the function is bindable and the binding parameter type is part of an inheritance hierarchy, the function overload is selected based on the type of the URL segment preceding the function name, which MAY be a type cast, see **[OData-URL]**.

11.5.3 Actions

Actions are operations exposed by an OData service that MAY have side effects when invoked. Actions MAY return data but MUST NOT be further composed.

11.5.3.1 Invoking an Action

To invoke an action bound to a resource, the client issues a `POST` request to an action URL. An action URL may be obtained from a previously returned entity instance or constructed by appending the namespace (or alias) qualified action name, optionally followed by open and close parenthesis `"()",` to a URL that identifies a resource whose type is the same as, or derives from, the type of the binding parameter of the action. The value for the binding parameter is the value of the resource identified by the URL prior to appending the action name, and any non-binding parameter values are passed in the request body according to the particular format.

To invoke an action through an action import, the client issues a `POST` request to a URL identifying the action import. The canonical URL for an action import is the service root, followed by the name of the action import, optionally followed by open and close parenthesis `"()"`. If the action import is defined in the default entity container, its unqualified name MAY be used, otherwise its namespace- (or alias-) and container-qualified name MUST be used. When invoking an action through an action import all non-default parameter values MUST be passed in the request body according to the particular format.

Any parameter values not specified in the request MUST be assumed to be the default value specified for the parameter, or null if no default is specified.

If the action returns results the client SHOULD use content type negotiation to request the results in the desired format, otherwise the default content type will be used.

If the client only wants an action to be processed when the binding parameter value, an entity or collection of entities, is unmodified, the client SHOULD include the `If-Match` header with the latest known ETag value for the entity or collection of entities.

On success, the response SHOULD be 200 for actions with a return type or 204 for action without a return type. The client can request whether any results from the action be returned using the Prefer header.

Example: The following JSON request invokes the `SampleEntities.CreateOrder` action using `/Customers('ALFKI')` as the customer (or binding parameter). The values `2` for the `quantity` parameter and `BLACKFRIDAY` for the `discountCode` parameter are passed in the body of the request:

```
POST http://server/Customers('ALFKI')/SampleEntities.CreateOrder

{
   "quantity": 2,
   "discountCode": "BLACKFRIDAY"
}
```

11.5.3.2 Action Overload Resolution

Actions support overloading by binding parameter, meaning a service MAY expose multiple actions with the same name that take a different binding parameter type.

If the action is bindable and the binding parameter type is part of an inheritance hierarchy, the action overload is selected based on the type of the URL segment preceding the action name, which MAY be a type cast, see **[OData-URL]**.

11.6 Batch Requests

Batch requests allow grouping multiple operations into a single HTTP request payload. A batch request MUST be represented as a Multipart MIME v1.0 message **[RFC2046]**, a standard format allowing the representation of multiple parts, each of which may have a different content type (as described in **[OData-Atom]** and **[OData-JSON]**), within a single request.

11.6.1 Batch Request Headers

Batch requests MUST be submitted as a single HTTP `POST` request to the batch endpoint of a service, which MUST be located at the URL `/$batch` relative to the Service Root URL.

The batch request MUST contain a `Content-Type` header specifying a content type of `multipart/mixed` and a boundary specification as defined in **[RFC2046]** is defined in the Batch Request Body section below.

```
POST /service/$batch HTTP/1.1
Host: odata.org
OData-Version: 1.0
OData-MaxVersion: 3.0
content-type: multipart/mixed; boundary=batch_36522ad7-fc75-4b56-8c71-56071383e77b

<Batch Request Body>
```

As shown in the example above, batch requests SHOULD contain applicable `OData-Version` headers.

11.6.2 Batch Request Body

The body of a batch request is be made up of a series of individual requests and change sets, each represented as a distinct MIME part (i.e. separated by the boundary defined in the `Content-Type` header).

The service MUST process the requests within a batch request sequentially, and MUST stop processing the request on the first error unless the `odata.continue-on-error` preference is specified.

An individual request in the context of a batch request is a Data request, Data Modification request, Action invocation request, or Function invocation request. A MIME part representing an individual request MUST include a `Content-Type` header with value `application/http` and a `Content-Transfer-Encoding` header with value `binary`.

Preambles and Epilogues in the MIME payload, as defined in **[RFC2046]**, are valid but are assigned no meaning and thus MUST be ignored by processors of batch requests.

The Request-URI of HTTP requests serialized within MIME part bodies MAY use one of the following three formats:

- Absolute URI with schema, host, port, and absolute resource path, for example

```
GET https://myserver.mydomain.org:1234/service/root/path/People(1) HTTP/1.1
```

- Absolute resource path and separate Host header, for example

```
GET /service/root/path/People(1) HTTP/1.1
Host: myserver.mydomain.org:1234
```

- Resource path relative to the batch request URI, for example

```
GET People(1) HTTP/1.1
```

Services MUST support all three formats.

Each MIME part body that represents a single request MUST NOT include:

- `authentication` or `authorization` related HTTP headers
- `Expect`, `From`, `Max-Forwards`, `Range`, or `TE` headers

Processors of batch requests MAY choose to disallow additional HTTP constructs in HTTP requests serialized within MIME part bodies. For example, a processor MAY choose to disallow chunked encoding to be used by such HTTP requests.

The example below shows a sample batch request that contains the following operations in the order listed:

1. A query request
2. Change Set that contains the following requests:
    1. Insert entity (with `Content-ID = 1`)
    2. Update request (with `Content-ID = 2`)
3. A second query request

```
POST /service/$batch HTTP/1.1
Host: host
OData-Version: 4.0
Content-Type: multipart/mixed; boundary=batch_36522ad7-fc75-4b56-8c71-56071383e77b

--batch_36522ad7-fc75-4b56-8c71-56071383e77b
Content-Type: application/http
Content-Transfer-Encoding:binary

GET /service/Customers('ALFKI')
Host: host
```

```
--batch_36522ad7-fc75-4b56-8c71-56071383e77b
Content-Type: multipart/mixed; boundary=changeset_77162fcd-b8da-41ac-a9f8-9357efbbd621
Content-Length: ###

--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd621
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 1

POST /service/Customers HTTP/1.1
Host: host
Content-Type: application/atom+xml;type=entry
Content-Length: ###

<AtomPub representation of a new Customer>

--changeset 77162fcd-b8da-41ac-a9f8-9357efbbd621
Content-Type: application/http
Content-Transfer-Encoding:binary
Content-ID: 2

PATCH /service/Customers('ALFKI') HTTP/1.1
Host: host
Content-Type: application/json
If-Match: xxxxx
Content-Length: ###

<JSON representation of Customer ALFKI>


--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd621--

--batch_36522ad7-fc75-4b56-8c71-56071383e77b
Content-Type: application/http
Content-Transfer-Encoding:binary

GET /service/Products HTTP/1.1
Host: host


--batch_36522ad7-fc75-4b56-8c71-56071383e77b--
```

Note: For brevity, in the example, request bodies are excluded in favor of English descriptions inside <> brackets and `OData-Version` headers are omitted.

11.6.3 Change Sets

A *change set* is an atomic unit of work consisting of an unordered group of one or more Data Modification requests or Action invocation requests. Change sets MUST NOT contain any `GET` requests or other change sets. The contents of a MIME part representing a change set MUST itself be a multipart MIME document (see **[RFC2046]**) with one part for each operation that makes up the change set. Each part representing an operation in the change set MUST include the same headers (`Content-Type` and `Content-Transfer-Encoding`) and associated values as previously described for query operations. In addition each request within a change set MUST specify a `Content-ID` header with a value unique within the batch request.

11.6.3.1 Referencing Requests in a Change Set

New entities created by an Insert request within a change set MAY be referenced by subsequent requests within the same change set by referring to the value of the `Content-ID` header prefixed with a `$` character. When used in this way, `$<Content-ID>` acts as an alias for the URI that identifies the new entity.

The example below shows a sample batch request that contains the following operations in the order listed:

A change set that contains the following requests:

1. Insert a new entity (with `Content-ID = 1`)
2. Insert a second new entity (references request with `Content-ID = 1`)

```
POST /service/$batch HTTP/1.1
Host: host
OData-Version: 4.0
Content-Type: multipart/mixed; boundary=batch 36522ad7-fc75-4b56-8c71-56071383e77b

--batch_36522ad7-fc75-4b56-8c71-56071383e77b
Content-Type: multipart/mixed; boundary=changeset_77162fcd-b8da-41ac-a9f8-9357efbbd621
Content-Length: ###

--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd621
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 1

POST /service/Customers HTTP/1.1
Host: host
Content-Type: application/atom+xml;type=entry
Content-Length: ###

<AtomPub representation of a new Customer>

--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd621
Content-Type: application/http
Content-Transfer-Encoding: binary

POST $1/Orders HTTP/1.1
Host: host
Content-Type: application/atom+xml;type=entry
Content-Length: ###

<AtomPub representation of a new Order>


--changeset 77162fcd-b8da-41ac-a9f8-9357efbbd621--


--batch_36522ad7-fc75-4b56-8c71-56071383e77b--
```

11.6.4 Responding to a Batch Request

Requests within a batch MUST be evaluated according to the same semantics used when the request appears outside the context of a batch.

The order of change sets and individual requests in a Batch request is significant as a service MUST process the components of the Batch in the order received. The order of requests within a change set is not significant as a service MAY process the requests within a change set in any order.

All operations in a change set represent a single change unit so a service MUST successfully process and apply all the requests in the change set or else apply none of them. It is up to the service implementation to define rollback semantics to undo any requests within a change set that may have been applied before another request in that same change set failed and thereby honor this all-or-nothing requirement. The service MAY execute the operations in a change set in any order and MAY return the responses to the individual requests in any order. The service MUST include the `Content-ID` header in each response with the same value that the client specified in the corresponding request, so clients can correlate requests and responses.

If the set of request headers of a Batch request are valid (the `Content-Type` is set to `multipart/mixed`, etc.) the service MUST return a `200 OK` HTTP response code to indicate that the request was accepted for processing, but the processing is yet to be completed. The requests within the body of the batch may subsequently fail or be malformed; however, this enables batch implementations to stream the results.

If the service receives a Batch request with an invalid set of headers it MUST return a `4xx response code` and perform no further processing of the request.

A response to a batch request MUST contain a `Content-Type` header with value `multipart/mixed`.

Structurally, a batch response body MUST match one-to-one with the corresponding batch request body, such that the same multipart MIME message structure defined for requests is used for responses. There are two exceptions to this rule. When a request within a change set fails, the change set response is not represented using the `multipart/mixed` media type. Instead, a single response, using the `application/http` media type and a `Content-Transfer-Encoding` header with a value of `binary`, is returned that applies to all requests in the change set and MUST be formatted according to the Error Handling defined for the particular response format. The other exception occurs when an error occurs processing a request and the `odata.continue-on-error` preference is not specified and therefore processing of the batch is terminated.

A response to a query operation in a batch MUST be formatted exactly as it would have appeared outside of a batch as described in Requesting Data or Invoking a Function, as appropriate.

For example, referencing the batch request example above, assume all the requests except the final query request succeed. In this case the response would be as shown in the following example.

```
HTTP/1.1 200 Ok
OData-Version: 4.0
Content-Length: ####
Content-Type: multipart/mixed; boundary=b_243234_25424_ef_892u748

--b_243234_25424_ef_892u748
Content-Type: application/http
Content-Transfer-Encoding: binary

HTTP/1.1 200 Ok
Content-Type: application/atom+xml;type=entry
Content-Length: ###

<AtomPub representation of the Customer entity with EntityKey ALFKI>

--b 243234 25424 ef 892u748
Content-Type: multipart/mixed; boundary=cs 12u7hdkin252452345eknd 383673037
Content-Length: ###

--cs_12u7hdkin252452345eknd_383673037
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 1

HTTP/1.1 201 Created
Content-Type: application/atom+xml;type=entry
Location: http://host/service.svc/Customer('POIUY')
Content-Length: ###

<AtomPub representation of a new Customer entity>
Content-ID: 2

--cs_12u7hdkin252452345eknd_383673037
Content-Type: application/http
Content-Transfer-Encoding: binary

HTTP/1.1 204 No Content
Host: host
```

```
--cs 12u7hdkin252452345eknd 383673037--

--b_243234_25424_ef_892u748
Content-Type: application/http
Content-Transfer-Encoding: binary

HTTP/1.1 404 Not Found
Content-Type: application/xml
Content-Length: ###

<Error message>


--b_243234_25424_ef_892u748--
```

11.6.5 Asynchronous Batch Requests

Batch requests may be executed asynchronously by including the `respond-async` preference in the Prefer header. The service MUST ignore the `respond-async` preference for individual responses within a batch.

After successful execution of the batch request the response to the batch request would be returned in the body of a response to an interrogation request against the monitor URL.

A service MAY return interim results to an asynchronously executing batch. While the server has more requests in the batch to process a response with a `202 Accepted` is added at the end of the list of responses to indicate the server is continuing to process the batch request. The client can use the monitor URL returned in this `202 Accepted` to continue processing the batch response.

For example, referencing the same batch request example above again, assume that when interrogating the monitor URL for the first time only the first request in the batch finished processing and all the remaining requests except the final query request succeed. In this case the responses would be as shown in the following example.

```
HTTP/1.1 200 Ok
OData-Version: 4.0
Content-Length: ####
Content-Type: multipart/mixed; boundary=b 243234 25424 ef 892u748

--b_243234_25424_ef_892u748
Content-Type: application/http
Content-Transfer-Encoding: binary

HTTP/1.1 200 Ok
Content-Type: application/atom+xml;type=entry
Content-Length: ###

<AtomPub representation of the Customer entity with EntityKey ALFKI>

--b_243234_25424_ef_892u748
Content-Type: application/http
Content-Transfer-Encoding: binary

HTTP/1.1 202 Accepted
Location: http://service-root/async-monitor
Retry-After: ###

--b_243234_25424_ef_892u748--
```

Client makes a second request using the returned monitor URL.

```
HTTP/1.1 200 Ok
OData-Version: 4.0
Content-Length: ####
Content-Type: multipart/mixed; boundary=b_243234_25424_ef_892u748

--b 243234 25424 ef 892u748
Content-Type: multipart/mixed; boundary=cs_12u7hdkin252452345eknd_383673037
Content-Length: ###

--cs_12u7hdkin252452345eknd_383673037
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 1

HTTP/1.1 201 Created
Content-Type: application/atom+xml;type=entry
Location: http://host/service.svc/Customer('POIUY')
Content-Length: ###

<AtomPub representation of a new Customer entity>
Content-ID: 2

--cs_12u7hdkin252452345eknd_383673037
Content-Type: application/http
Content-Transfer-Encoding: binary

HTTP/1.1 204 No Content
Host: host


--cs 12u7hdkin252452345eknd 383673037--

--b_243234_25424_ef_892u748
Content-Type: application/http
Content-Transfer-Encoding: binary

HTTP/1.1 404 Not Found
Content-Type: application/xml
Content-Length: ###

<Error message>


--b_243234_25424_ef_892u748—
```

Since a change set is executed atomically, `202 Accepted` MUST NOT be returned within a change set.
`202 Accepted`

12 Conformance

OData is designed as a set of conventions that can be layered on top of existing standards to provide common representations for common functionality. Not all services will support all of the conventions defined in the protocol; services choose those conventions defined in OData as the representation to expose that functionality appropriate for their scenarios.

To aid in client/server interoperability, this specification defines multiple levels of conformance for an OData Service, as well as the minimal requirements for an OData Client to be interoperable across OData services.

12.1 OData Service Conformance Levels

OData defines three levels of conformance for an OData Service.

The conformance levels are design to correspond to different service scenarios. For example, a service that publishes data compliant with one or more of the OData defined formats may comply with the OData Minimal Conformance Level without supporting any additional functionality. A service that offers more control over the data that the client retrieves may comply with the OData Intermediate Conformance Level. Services that conform to the OData Advanced Conformance Level can expect to interoperate with the most functionality against the broadest range of generic clients.

Services are encouraged to support as much additional functionality beyond their level of conformance as is appropriate for their intended scenario.

12.1.1 OData Minimal Conformance Level

In order to conform to the OData Minimal conformance level, a service:

- MUST publish a service document at the service root
- MUST return data according to at least one of the OData defined formats
- MUST support server-driven paging when returning partial results
- MUST return the appropriate `OData-Version` header
- MUST honor the semantics the following headers, or fail the request
  - `Accept`
  - `OData-MaxVersion`
- MUST follow OData guidelines for extensibility
- MUST successfully parse the request according to **[OData-ABNF]** for any supported system query string options and either follow the specification or return `501 Not Implemented` for any unsupported functionality
- MUST expose only data types defined in **[OData-CSDL]**
- MUST NOT require clients to understand any metadata or instance annotations, custom headers, or custom content in the payload in order to correctly consume the service
- MUST NOT violate any OData update semantics
- SHOULD support $expand specifically to enable clients to define the extent of the graph to retrieve
- MUST NOT violate any other OData defined semantics

In addition, to be considered an *Updatable OData Service*, the service:

- MUST include edit links (explicitly or implicitly) for all updatable or deletable resources
- MUST support POST of new entities to insertable entity sets
- MUST support POST of new related entities to updatable navigation properties
- MUST support POST to `$ref` to add an existing entity to an updatable related collection
- MUST support PUT to `$ref` to set an existing single updatable related entity
- MUST support PATCH to all edit links for updatable resources
- MUST support DELETE to all edit links for deleteable resources

- MUST support DELETE to `$ref` to remove an entity from an updatable navigation property
- MUST support if-match header in update/delete of any resources returned with an ETag
- MUST return a `Location` header from POST with the edit-link or self-link of the created resource
- MUST include the `OData-EntityId` header in response to any POST/PATCH that returns `204 No Content`
- MUST support Upserts
- SHOULD support PUT and PATCH to an individual primitive or complex property (respectively)
- SHOULD support DELETE to set an individual property to null
- SHOULD support deep inserts

12.1.2 OData Intermediate Conformance Level

In order to conform to the OData Intermediate Conformance level, a service:

- MUST conform to the OData Minimal level
- SHOULD publish metadata at `/$metadata` according to **[OData-CSDL]**
- SHOULD support the OData JSON format
- MUST successfully parse the **[OData ABNF]** and either follow the specification or return `501 Not Implemented` for any unsupported functionality
- MUST support `$select`
- MUST support casting to a derived type if present in the model
- MUST support `$top`
- MUST support `/$value` on media entities and individual properties
- MUST support `$filter`
    - MUST support `eq`, `ne` filter operations on filterable properties
    - MUST support aliases in `$filter` expressions
    - SHOULD support additional filter operations and MUST fail on any unsupported filter operations
    - SHOULD support the canonical functions and MUST fail on any unsupported canonical functions
- SHOULD support `$search`
- SHOULD support `$skip`
- SHOULD support `/$count`
- SHOULD support `$expand` for expandable navigation properties
    - SHOULD support `$filter` on expanded properties
- SHOULD support the canonical functions `any` and `all` on navigation and collection-valued properties
- SHOULD support `/$count` on navigation, collection properties within a `$filter` expression
- SHOULD support `$orderby asc` and `desc` on individual properties

12.1.3 OData Advanced Conformance Level

In order to conform to the OData Advanced Conformance level, a service:

- MUST conform to at least the OData Intermediate Conformance Level
- MUST publish metadata at `/$metadata` according to [OData-CSDL]
- MUST support the OData JSON format
- MUST support `/$count` on navigation and collection properties within a `$filter` expression
- MUST support the canonical functions `any` and `all` on navigation and collection-valued properties
- MUST support `$skip`
- MUST support `/$count` on collection valued resource paths
- MUST support the `$count` system query option
- MUST support `$orderby asc` and `desc` on individual properties
- MUST support `$expand` for expandable navigation properties

    - MUST support returning references for expanded properties
    - MUST support `$filter` on filterable properties of expanded entities
    - MUST support cast segment in expand with derived types
    - SHOULD support `$orderby` `asc` and `desc` on individual properties
    - SHOULD support the `$count` system query option for expanded properties
    - SHOULD support `$top` and `$skip` on expanded properties
    - SHOULD support `$search` on expanded properties
    - SHOULD support `$levels` for recursive expand
- MUST support `$search`
- MUST support `$batch`
- MUST support the resource path conventions defined in **[OData-URL]**
- SHOULD support Asynchronous operations
- SHOULD support Delta change tracking
- SHOULD support queries rooted at the entity container
- SHOULD support a conforming OData service interface over metadata

12.2 Interoperable OData Clients

Interoperable OData Clients can expect to work with OData Services that comply with at least the OData Minimal conformance level and implement the **[OData-JSON]** format. Clients that additionally support **[OData-Atom]** can expect to interoperate with a broader range of OData Services.

To be generally interoperable, OData Clients

- MUST specify the `Odata-MaxVersion` header in request
- MUST specify `OData-Version` and `Content-Type` in any request with a payload
- MUST be a conforming consumer of OData as defined in **[OData-JSON]**
- MUST follow redirects
- MUST correctly handle next links
- MUST be a conformant client of metadata as defined in **[OData-CSDL]**
- MUST support instances returning properties and navigation properties not specified in metadata
- MUST support entity instances with external metadata
- MUST support properties with externally defined data types
- MAY request entity references in place of entities previously returned in the response
- MAY support deleted entries, link entries, deleted link entries in a delta response
- MAY support asynchronous responses
- MAY support `odata.metadata=minimal` in a JSON response
- MAY support `odata.streaming`

To support updates, OData Clients:

- MUST generate PATCH requests for updates

Appendix A. Acknowledgments

The following individuals were members of the OASIS OData Technical Committee during the creation of this specification and their contributions are gratefully acknowledged:

- Howard Abrams (CA Technologies)
- Ken Baclawski (Northeastern University)
- Jay Balunas (Red Hat)
- Mark Biamonte (Progress Software)
- Matthew Borges (SAP AG)
- Edmond Bourne (BlackBerry)
- Joseph Boyle (Planetwork, Inc.)
- Peter Brown (Individual)
- Antonio Campanile (Bank of America)
- Pablo Castro (Microsoft)
- Axel Conrad (BlackBerry)
- Robin Cover (OASIS)
- Erik de Voogd (SDL)
- Diane Downie (Citrix Systems)
- Stefan Drees (Individual)
- Patrick Durusau (Individual)
- Andrew Eisenberg (IBM)
- Chet Ensign (OASIS)
- Davina Erasmus (SDL)
- Colleen Evans (Microsoft)
- Senaka Fernando (WSO2)
- Brent Gross (IBM)
- Zhun Guo (Individual)
- Anila Kumar GVN (CA Technologies)
- Ralf Handl (SAP AG)
- Barbara Hartel (SAP AG)
- Hubert Heijkers (IBM)
- Jens Hüsken (SAP AG)
- Evan Ireland (SAP AG)
- Gershon Janssen (Individual)
- Ram Jeyaraman (Microsoft)
- Ted Jones (Red Hat)
- Diane Jordan (IBM)
- Stephan Klevenz (SAP AG)
- Gerald Krause (SAP AG)
- Nuno Linhares (SDL)
- Paul Lipton (CA Technologies)
- Susan Malaika (IBM)
- Ramanjaneyulu Malisetti (CA Technologies)
- Neil McEvoy (iFOSSF – International Free and Open Source Solutions Foundation)
- Stan Mitranic (CA Technologies)
- Dale Moberg (Axway Software)
- Graham Moore (BrightstarDB Ltd.)
- Farrukh Najmi (Individual)
- Shishir Pardikar (Citrix Systems)
- Sanjay Patil (SAP AG)
- Nuccio Piscopo (iFOSSF – International Free and Open Source Solutions Foundation)

- Michael Pizzo (Microsoft)
- Robert Richards (Mashery)
- Sumedha Rubasinghe (WSO2)
- James Snell (IBM)
- Jeffrey Turpin (Axway Software)
- John Willson (Individual)
- John Wilmes (Individual)
- Christopher Woodruff (Perficient, Inc.)
- Martin Zurmuehl (SAP AG)

Appendix B. Revision History

Revision	Date	Editor	Changes Made
Working Draft 01	2012-08-22	Michael Pizzo	Translated Contribution to OASIS format/template
Committee Specification Draft 01	2013-04-26	Michael Pizzo, Ralf Handl, Martin Zurmuehl	Added Delta support, Asynchronous processing, Upsert Aligned and expanded Prefer header preferences Simplified data model Defined rules and semantics around distributed metadata Fleshed out descriptions and examples and addressed numerous editorial and technical issues processed through the TC Added Conformance section