OData Version 4.02. Part 1: Protocol

Committee Specification Draft 02

28 February 2024

This stage:

https://docs.oasis-open.org/odata/odata/v4.02/csd02/part1-protocol/odata-v4.02-csd02-part1-protocol.md (Authoritative)
https://docs.oasis-open.org/odata/odata/v4.02/csd02/part1-protocol/odata-v4.02-csd02-part1-protocol.html
https://docs.oasis-open.org/odata/odata/v4.02/csd02/part1-protocol/odata-v4.02-csd02-part1-protocol.pdf

Previous stage:

https://docs.oasis-open.org/odata/odata/v4.02/csd01/part1-protocol/odata-v4.02-csd01-part1-protocol.md (Authoritative)
https://docs.oasis-open.org/odata/odata/v4.02/csd01/part1-protocol/odata-v4.02-csd01-part1-protocol.html
https://docs.oasis-open.org/odata/odata/v4.02/csd01/part1-protocol/odata-v4.02-csd01-part1-protocol.pdf

Latest stage:

https://docs.oasis-open.org/odata/odata/v4.02/odata-v4.02-part1-protocol.md (Authoritative)
https://docs.oasis-open.org/odata/odata/v4.02/odata-v4.02-part1-protocol.html
https://docs.oasis-open.org/odata/odata/v4.02/odata-v4.02-part1-protocol.pdf

Technical Committee:

OASIS Open Data Protocol (OData) TC

Chairs:

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

Editors:

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

Additional artifacts:

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

- *OData Version 4.02 Part 1: Protocol* (this document). https://docs.oasis-open.org/odata/odata/v4.02/csd02/part1-protocol/odata-v4.02-csd02-part1-protocol.html
- *OData Version 4.02 Part 2: URL Conventions*. https://docs.oasis-open.org/odata/odata/v4.02/csd02/part2-url-conventions/odata-v4.02-csd02-part2-url-conventions.html

- ABNF components: *OData ABNF Construction Rules Version 4.02 and OData ABNF Test Cases Version 4.02*. https://docs.oasis-open.org/odata/odata/v4.02/csd02/abnf/.

Related work:

This specification replaces or supersedes:

- *OData Version 4.01. Part 1: Protocol*. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. OASIS Standard. Latest stage: https://docs.oasis-open.org/odata/odata/v4.01/odata-v4.01-part1-protocol.html
- *OData Version 4.0. Part 1: Protocol*. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. OASIS Standard. Latest stage: http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part1-protocol.html

This specification is related to:

- *OData Vocabularies Version 4.0*. Edited by Michael Pizzo, Ralf Handl, and Ram Jeyaraman. Latest stage: https://docs.oasis-open.org/odata/odata-vocabularies/v4.0/odata-vocabularies-v4.0.html
- *OData Common Schema Definition Language (CSDL) JSON Representation Version 4.02*. Edited by Michael Pizzo, Ralf Handl, and Heiko Theißen. Latest stage: https://docs.oasis-open.org/odata/odata-csdl-json/v4.02/odata-csdl-json-v4.02.html
- *OData Common Schema Definition Language (CSDL) XML Representation Version 4.02*. Edited by Michael Pizzo, Ralf Handl, and Heiko Theißen. Latest stage: https://docs.oasis-open.org/odata/odata-csdl-xml/v4.02/odata-csdl-xml-v4.02.html
- *OData JSON Format Version 4.02*. Edited by Michael Pizzo, Ralf Handl, and Heiko Theißen. Latest stage: https://docs.oasis-open.org/odata/odata-json-format/v4.02/odata-json-format-v4.02.html
- *OData Data Aggregation Extension Version 4.0*. Edited by Ralf Handl, Hubert Heijkers, Gerald Krause, Michael Pizzo, Heiko Theißen, and Martin Zurmuehl. Latest stage: https://docs.oasis-open.org/odata/odata-data-aggregation-ext/v4.0/odata-data-aggregation-ext-v4.0.html
- *OData Extension for Temporal Data Version 4.0*. Edited by Ralf Handl, Hubert Heijkers, Gerald Krause, Michael Pizzo, Heiko Theißen, and Martin Zurmuehl. Latest stage: https://docs.oasis-open.org/odata/odata-temporal-ext/v4.0/odata-temporal-ext-v4.0.html

Abstract:

The Open Data Protocol (OData) enables the creation of REST-based data services, which allow resources, identified using Uniform Resource Locators (URLs) 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 stage” location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=odata#technical.

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

This specification is provided under the RF on RAND Terms Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. 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 TC’s web page (https://www.oasis-open.org/committees/odata/ipr.php).

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

Key words:

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

Citation format:

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

[OData-v4.02-Part1]

OData Version 4.02. Part 1: Protocol. Edited by Michael Pizzo, Ralf Handl, and Heiko Theißen. 28 February 2024. OASIS Committee Specification Draft 02. https://docs.oasis-open.org/odata/odata/v4.02/csd02/odata-v4.02-csd02-part1-protocol.html. Latest stage: https://docs.oasis-open.org/odata/odata/v4.02/odata-v4.02-part1-protocol.html.

Notices

Table of Contents

1 Introduction

The Open Data Protocol (OData) enables the creation of REST-based data services which allow resources, identified using Uniform Resource Locators (URLs) and defined in a data model, to be published and edited by Web clients using simple HTTP messages. This specification defines the core semantics and the behavioral aspects of the protocol.

The **[OData-URL]** specification defines a set of rules for constructing URLs to identify the data and metadata exposed by an OData service as well as a set of reserved URL query options.

The **[OData-CSDLJSON]** specification defines a JSON representation of the entity data model exposed by an OData service.

The **[OData-CSDLXML]** specification defines an XML representation of the entity data model exposed by an OData service.

The **[OData-JSON]** document specifies the JSON format of the resource representations that are exchanged using OData.

1.1 Changes from Earlier Versions

Section	Feature / Change	Issue
Section 10.2	Context URLs use parentheses-style keys without percent-encoding	368
Section 11.4	Response code `204 No Content` after successful data modification if requested response could not be constructed	443
Section 11.4.4	Upserts to single-valued non-containment navigation properties	455
Section 11.4.9.3	Setting a complex property to a different type	534
Section 12	Allow `400 Bad Request` in addition to `501 Not Implemented` for unsupported functionality	391
Section 12.3	Encoding of plus character in URLs	485

1.2 Glossary

1.2.1 Definitions of Terms

1.2.2 Acronyms and Abbreviations

1.2.3 Document Conventions

Keywords defined by this specification use `this monospaced font`.

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

Example 1: text describing an example uses this paragraph style

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

All examples in this document are non-normative and informative only.

All other text is normative unless otherwise labeled.

2 Overview

The OData Protocol is an application-level protocol for interacting with data via RESTful interfaces. 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 service.
- 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 sources. 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.
- 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 is 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, actions, and functions.

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

Entity types are named structured types with a key. They define the named properties and relationships of an entity. Entity types may derive by single inheritance from other entity types.

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

Complex types are keyless named structured types consisting of a set of properties. These are value types whose instances cannot be referenced outside of their containing entity. 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 cannot have the same name as a declared property. Entity or complex types which allow clients to persist additional undeclared properties are called *open 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 with 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 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. If multiple entity sets use the same entity type, the same combination of key values can appear in more than one entity set and identifies different entities, one per entity set where this key combination appears. Each of these entities has a different entity-id. 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 may support further composition, for example, with additional filter operations, functions or an action. *Actions* are operations that allow side effects, such as data modification, and cannot be further composed in order to avoid non-deterministic behavior. Actions and functions are either *bound* to a type, enabling them to be called as members of an instance of that type, or unbound, in which case they are called as static operations. *Action imports* and *function imports* enable unbound actions and functions to be called from the service root.

Singletons are named entities which can be accessed as direct children of the entity container. A singleton may also be a member of an entity set.

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

Refer to **[OData-CSDLJSON]** or **[OData-CSDLXML]** 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 singletons 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 Entity References

Whereas entities within an entity set are uniquely identified by their key values, entities are also uniquely identified by a durable, opaque, globally unique *entity-id*. The entity-id MUST be an IRI as defined in **[RFC3987]** and MAY be expressed in payloads, headers, and URLs as a relative reference as appropriate. While the client MUST be prepared to accept any IRI, services MUST use valid URIs in this version of the specification since there is currently no lossless representation of an IRI in the `EntityId` header.

Services are strongly encouraged to use the canonical URL for an entity as defined in **OData-URL** as its entity-id, but clients cannot assume the entity-id can be used to locate the entity unless the `Core.DereferenceableIDs` term is applied to the entity container, nor can the client assume any semantics from the structure of the entity-id. The canonical resource `$entity` provides a general mechanism for resolving an entity-id into an entity representation.

Services that use the standard URL conventions for entity-ids annotate their entity container with the term `Core.ConventionalIDs`, see **[OData-VocCore]**.

Entity references refer to an entity using the entity’s entity-id.

4.2 Read URLs and Edit URLs

The read URL of an entity is the URL that can be used to read the entity.

The edit URL of an entity is the URL that can be used to update or delete the entity.

The edit URL of a property is the edit URL of the entity with appended segment(s) containing the path to the property.

Services are strongly encouraged to use the canonical URL for an entity as defined in **OData-URL** for both the read URL and the edit URL of an entity, with a cast segment to the type of the entity appended to the canonical URL if the type of the entity is derived from the declared type of the entity set. However, clients cannot assume this convention and must use the links specified in the payload according to the appropriate format as the two URLs may be different from one another, or one or both of them may differ from convention.

4.3 Transient Entities

Transient entities are instances of an entity type that are dynamically generated on request and only exist within a response payload. They do not possess an entity-id or an update URL and consequently cannot be updated. A transient entity may have a read URL, which generates a new transient entity using the same algorithm.

4.4 Default Namespaces

References to actions, functions, and types within a URL typically requires prefixing the name of the action, function, or type with the namespace or alias in which it is defined. This namespace qualification enables differentiating between similarly named elements across schema, or between properties and bound functions, actions, or types with the same name.

Services MAY define one or more default namespaces through the `Core.DefaultNamespace` term defined in **[OData-VocCore]**. Functions, actions and types in a default namespace can be referenced in URLs with or without namespace or alias qualification.

Service designers should ensure uniqueness of schema children across all default namespaces, and should avoid naming bound functions, actions, or derived types with the same name as a structural or navigation property of the type.

In the case where ambiguity does exist, an unqualified segment appended to a structured value is always first compared to the list of properties defined on the structured type. If no defined property with a name matching the unqualified segment exists, or the preceding segment represents a collection or a scalar value, it is next compared to the names of any bound functions or actions, or derived type names, defined within any default namespace. If it still does not match, and the preceding segment represents a structured value, it is interpreted as a dynamic property.

Services MAY disallow dynamic properties on structured values whose names conflict with a bound action, function, or derived type defined within in a default namespace.

The behavior if name conflicts occur across children of default namespaces is undefined. Generic clients are encouraged to always qualify action, function, and type names in order to avoid any possible ambiguity.

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 are versioned according to the `OData-Version` header.

OData clients include the `OData-MaxVersion` header in requests in order to specify the maximum acceptable response version. Services respond with the maximum supported version that is less than or equal to the requested `OData-MaxVersion`, using decimal comparison. The syntax of the `OData-Version` and `OData-MaxVersion` header fields is defined in **[OData-ABNF]**.

Services SHOULD advertise supported versions of OData through the `Core.ODataVersions` term, defined in **[OData-VocCore]**.

This version of the specification defines OData version values `4.0` and `4.01`. Content that applies only to one version or another is explicitly called out in the text.

5.2 Model Versioning

The Data Model exposed by an OData Service defines a contract between the OData service and its clients. Services are allowed to extend their model only to the degree that it does not break existing clients. Breaking changes, such as removing properties, changing the type of existing properties, adding or removing key properties, or reordering action or function parameters, require that a new service version is provided at a different service root URL for the new model, or that the service version its metadata using the `Core.SchemaVersion` annotation, defined in **[OData-VocCore]**.

Services that version their metadata MUST support version-specific requests according to the `$schemaversion` system query option. The following Data Model additions are considered safe and do not require services to version their entry point or schema.

- Adding a property that is nullable or has a default value; if it has the same name as an existing dynamic property, it must have the same type (or base type) as the existing dynamic property
- Adding a navigation property that is nullable or collection-valued; if it has the same name as an existing dynamic navigation property, it must have the same type (or base type) as the existing dynamic navigation property
- Adding a new entity type to the model
- Adding a new complex type to the model
- Adding a new entity set
- Adding a new singleton
- Adding an action, a function, an action import, or function import
- Adding an action parameter that is nullable after existing parameters
- Adding an action or function parameter that is annotated with `Core.OptionalParameter` after existing parameters
- Adding a type definition or enumeration
- Adding a new term
- 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 and derived types not previously defined by the service.

Services SHOULD NOT change their data model depending on the authenticated user. If the data model is user or user-group dependent, all changes MUST be *safe changes* as defined in this section when comparing the full model to the model visible to users with restricted authorizations.

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 optionally prefixed with "`$`". Services may support additional custom query options not defined in the OData specification, but they MUST NOT begin with the "`$`" or "`@`" character and MUST NOT conflict with any OData-defined system query options defined in the OData version supported by the service.

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 do not understand and MUST fail any request that contains unsupported OData query options defined in the version of this specification supported by the service.

In many cases OData services return URLs to identify resources that are later requested by clients. Where possible, interoperability is enhanced by providing all identifying information in the path portion of the URL. However, clients should be prepared for such URLs to include custom query options and propagate any such custom query options in future requests to the identified resource.

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 can 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 include a namespace or alias prefix. The `Edm`, `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

The set of annotations defined within a schema comprise a *vocabulary*. Shared vocabularies provide 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 conform to the processing requirements for the headers defined by this specification for that version.

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]** 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 defined in **[RFC9110]**, or through the system query option `$format`.

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

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

Services SHOULD advertise their supported formats in the metadata document by annotating their entity container with the term `Capabilities.SupportedFormats`, as defined in **[OData-VocCap]**, listing all available formats and combinations of supported format parameters.

The media types for the JSON and XML representation of the metadata document are described in section "Metadata Document Request".

The format specification **[OData-JSON]** describes the media type and the format parameters for non-metadata requests and responses.

For non-metadata requests, if neither the `Accept` header nor the `$format` query option are specified, the service MAY respond to requests in any format.

Client libraries MUST retain the order of objects within an array in JSON responses, and elements in document order for XML responses, including CSDL documents.

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 following headers are common between OData requests and responses.

8.1.1 Header Content-Type

The format of a non-empty individual request or response body, alone or within a batch, MUST be specified in the `Content-Type` header of a request or response. The exception to this is if the body represents the media stream of a media entity or stream property, in which case the `Content-Type` header SHOULD be present.

The specified format MAY include format parameters. Clients MUST be prepared for the service to return custom format parameters not defined in OData and SHOULD NOT expect that such format parameters can be ignored. Custom format parameters MUST NOT start with `odata` and services MUST NOT require generic OData consumers to understand custom format parameters in order to correctly interpret the payload.

See **[OData-JSON]** for format-specific details about format parameters within the `Content-Type` header.

8.1.2 Header Content-Encoding

As defined in **[RFC9110]**, the `Content-Encoding` header field is used as a modifier to the media-type (as indicated in the `Content-Type` header). When present, its value indicates what additional content codings have been applied to the entity-body. A service MAY specify a list of acceptable content codings using an annotation with term `Capabilities.AcceptableEncodings`, see **[OData-VocCap]**.

If the `Content-Encoding` header is specified on an individual request or response within a batch, then it specifies the encoding for that individual request or response. Individual requests or responses that don't include the `Content-Encoding` header inherit the encoding of the overall batch request or response.

8.1.3 Header Content-Language

As defined in **[RFC9110]**, a request or response can include a `Content-Language` header to indicate the natural language of the intended audience for the enclosed message body. OData does not add any additional requirements over HTTP for including `Content-Language`. OData services can annotate model elements whose content depends on the content language with the term `Core.IsLanguageDependent`, see **[OData-VocCore]**.

If the `Content-Language` header is specified on an individual request or response within a batch, then it specifies the language for that individual request or response. Individual requests or responses that don't include the `Content-Language` header inherit the language of the overall batch request or response.

8.1.4 Header Content-Length

As defined in **[RFC9110]**, 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`.

If the `Content-Length` header is specified on an individual request or response within a batch, then it specifies the length for that individual request or response.

8.1.5 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 payload.

If present on a request, the service MUST interpret the request payload 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 payload is generated using the minimum 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 payload. The client MUST interpret the response payload according to the rules defined in the specified version of the protocol. Request and response payloads are independent and may have different `OData-Version` headers according to the above rules.

For more details, see Versioning.

If the `OData-Version` header is specified on an individual request or response within a batch, then it specifies the OData version for that individual request or response. Individual requests or responses that don't include the `OData-Version` header inherit the OData version of the overall batch request or response. This OData version does not typically vary within a batch.

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 defined in **[RFC9110]**, the client MAY specify the set of accepted formats with the `Accept` Header.

Services MUST reject formats that specify unknown or unsupported format parameters.

If a 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.

The service SHOULD NOT add any format parameters to the `Content-Type` header not specified in the `Accept` header.

If the `Accept` header is specified on an individual request within a batch, then it specifies the acceptable formats for that individual request. Requests within a batch that don't include the `Accept` header inherit the acceptable formats of the overall batch request.

8.2.2 Header `Accept-Charset`

As defined in **[RFC9110]**, the client MAY specify the set of accepted character sets with the `Accept-Charset` header.

If the `Accept-Charset` header is specified on an individual request within a batch, then it specifies the acceptable character sets for that individual request. Requests within a batch that don't include the `Accept-Charset` header inherit the acceptable character sets of the overall batch request.

8.2.3 Header `Accept-Language`

As defined in **[RFC9110]**, the client MAY specify the set of accepted natural languages with the `Accept-Language` header.

If the `Accept-Language` header is specified on an individual request within a batch, then it specifies the acceptable languages for that individual request. Requests within a batch that don't include the `Accept-Language` header inherit the acceptable languages of the overall batch request.

8.2.4 Header If-Match

As defined in **[RFC9110]**, a client MAY include an `If-Match` header in a request to `GET`, `POST`, `PUT`, `PATCH` or `DELETE`. The value of the `If-Match` request header MUST be an ETag value previously retrieved for the resource, or `*` to match any value.

If an operation on an existing resource requires an ETag, (see term `Core.OptimisticConcurrency` in **[OData-VocCore]** and property `OptimisticConcurrencyControl` of type `Capabilities.NavigationPropertyRestriction` in **[OData-VocCap]**) and the client does not specify an `If-Match` request header in a Data Modification Request or in an Action Request invoking an action bound to the resource, the service responds with a `428 Precondition Required` and MUST ensure that no observable change occurs as a result of the request.

If present, the request MUST only be processed if the specified ETag value matches the current ETag value of the target resource. Services sending `ETag` headers with weak ETags that only depend on the representation-independent entity state MUST use the weak comparison function because it is sufficient to prevent accidental overwrites. This is a deviation from **[RFC9110]**.

If the value does not match the current ETag value of the resource 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.

An `If-Match` header with a value of `*` in a `PUT` or `PATCH` request results in an upsert request being processed as an update and not an insert.

The `If-Match` header MUST NOT be specified on a batch request, but MAY be specified on individual requests within the batch.

8.2.5 Header If-None-Match

As defined in **[RFC9110]**, a client MAY include an `If-None-Match` header in a request to `GET`, `POST`, `PUT`, `PATCH` or `DELETE`. The value of the `If-None-Match` request header MUST be an ETag value previously retrieved for the resource, or `*`.

If present, the request MUST only be processed if the specified ETag value does not match the current ETag value of the resource, using the weak comparison function (see **[RFC9110]**). If the value matches the current ETag value of the resource, 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.

An `If-None-Match` header with a value of `*` in a `PUT` or `PATCH` request results in an upsert request being processed as an insert and not an update.

The `If-None-Match` header MUST NOT be specified on a batch request, but MAY be specified on individual requests within the batch.

8.2.6 Header Isolation (OData-Isolation)

The `Isolation` header specifies the isolation of the current request from external changes. The only supported value for this header is `snapshot`.

If the service doesn't support `Isolation:snapshot` and this header was specified on the request, the service MUST NOT process the request and MUST respond with `412 Precondition Failed`.

Snapshot isolation guarantees that all data returned for a request, including multiple requests within a batch or results retrieved across multiple pages, will be consistent as of a single point in time. Only data modifications made within the request (for example, by a data modification request within the same batch) are visible. The effect is as if the request generates a "snapshot" of the committed data as it existed at the start of the request.

The `Isolation` header may be specified on a single or batch request. If it is specified on a batch then the value is applied to all statements within the batch.

Next links returned within a snapshot return results within the same snapshot as the initial request; the client is not required to repeat the header on each individual page request.

The `Isolation` header has no effect on links other than the next link. Navigation links, read links, and edit links return the current version of the data.

A service returns `410 Gone` or `404 Not Found` if a consumer tries to follow a next link referring to a snapshot that is no longer available.

The syntax of the `Isolation` header is defined in **[OData-ABNF]**.

A service MAY specify the support for `Isolation:snapshot` using an annotation with term `Capabilities.IsolationSupported`, see **[OData-VocCap]**.

Note: The `Isolation` header was named `OData-Isolation` in OData version 4.0. Services that support the `Isolation` header SHOULD also support `OData-Isolation` for OData 4.0 clients and clients SHOULD use `OData-Isolation` for compatibility with OData 4.0 services. If both `Isolation` and `OData-Isolation` headers are specified in the same request, the value of the `Isolation` header SHOULD be used.

8.2.7 Header `OData-MaxVersion`

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

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

If `OData-MaxVersion` is not specified, then the service SHOULD return responses with the same OData version over time and interpret the request as having an `OData-MaxVersion` equal to the maximum OData version supported by the service at its initial publication.

If the `OData-MaxVersion` header is specified on an individual request within a batch, then it specifies the maximum OData version for that individual request. Individual requests that don't include the `OData-MaxVersion` header inherit the maximum OData version of the overall batch request or response. The maximum OData version does not typically vary within a batch.

For more details, see Versioning.

8.2.8 Header `Prefer`

The `Prefer` header, as defined in **[RFC7240]**, allows clients to request certain behavior from the service. The service MUST ignore preference values that are either not supported or not known 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` and `Vary` headers.

8.2.8.1 Preference `allow-entityreferences` (`odata.allow-entityreferences`)

The `allow-entityreferences` preference indicates that the service is allowed to 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 `allow-entityreferences` has not been specified in the request, unless

explicitly defined by other rules in this document. The syntax of the `allow-entityreferences` preference is defined in **[OData-ABNF]**.

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

If the `allow-entityreferences` preference is specified on an individual request within a batch, then it specifies the preference for that individual request. Individual requests within a batch that don't include the `allow-entityreferences` preference inherit the preference of the overall batch request.

Note: The `allow-entityreferences` preference was named `odata.allow-entityreferences` in OData version 4.0. Services that support the `allow-entityreferences` preference SHOULD also support `odata.allow-entityreferences` for OData 4.0 clients and clients SHOULD use `odata.allow-entityreferences` for compatibility with OData 4.0 services.

8.2.8.2 Preference `callback` (`odata.callback`)

For scenarios in which links returned by the service are used by the client to poll for additional information, the client can specify the `callback` preference to request that the service notify the client when data is available.

The `callback` preference can be specified:

- when requesting asynchronous processing of a request with the `respond-async` preference, or
- on a `GET` request to a delta link.

The `callback` preference MUST include the parameter `url` whose value is the URL of a callback endpoint to be invoked by the OData service when data is available. The syntax of the `callback` preference is defined in **[OData-ABNF]**.

For HTTP based callbacks, the OData service executes an HTTP `GET` request against the specified URL.

Services that support `callback` SHOULD support notifying the client through HTTP. Services can advertise callback support using the `Capabilities.CallbackSupported` annotation term defined in **[OData-VocCap]**.

If the service applies the `callback` preference it MUST include the `callback` preference in the `Preference-Applied` response header.

When the `callback` preference is applied to asynchronous requests, the OData service invokes the callback endpoint once it has finished processing the request. The status monitor resource, returned in the `Location` header of the previously returned `202 Accepted` response, can then be used to retrieve the results of the asynchronously executed request.

When the `callback` preference is specified on a `GET` request to a delta link and there are no changes available, the OData service returns a `202 Accepted` response with a `Location` header specifying the delta link to be used to check for future updates. The OData service then invokes the specified callback endpoint once new changes become available.

Combining `respond-async`, `callback` and `track-changes` preferences on a `GET` request to a delta-link might influence the response in a couple of ways.

- If the service processes the request synchronously, and no updates are available, then the response is the same as if the respond-async hadn't been specified and results in a response as described above.
- If the service processes the request asynchronously, then it responds with a `202 Accepted` response specifying the URL to the status monitor resource as it would have with any other asynchronous request. Once the service has finished processing the asynchronous request to the delta link resource, if changes are available it invokes the specified callback endpoint. If no changes are available, the service SHOULD wait to notify the client until changes are available. Once notified, the client uses the status monitor resource from the Location header of

the previously returned 202 Accepted response to retrieve the results. In case no updates were available after processing the initial request, the result will contain no updates and the client can use the delta-link contained in the result to retrieve the updates that have since become available.

If the consumer specifies the same URL as callback endpoint in multiple requests, the service MAY collate them into a single notification once additional data is available for any of the requests. However, the consumer MUST be prepared to deal with receiving up to as many notifications as it requested.

Example 2: using a HTTP callback endpoint to receive notification

```
Prefer: callback; url="http://myserver/notfication/token/12345"
```

If the `callback` preference is specified on an individual request within a batch, then it specifies the callback to be used for tracking changes to that individual request. If the `callback` preference is specified on a batch, then it specifies the callback to be used for async responses to the batch.

Note: The `callback` preference was named `odata.callback` in OData version 4.0. Services that support the `callback` preference SHOULD also support `odata.callback` for OData 4.0 clients and clients SHOULD use `odata.callback` for compatibility with OData 4.0 services. If both `callback` and `odata.callback` preferences are specified in the same request, the value of the `callback` preference SHOULD be used.

8.2.8.3 Preference `continue-on-error` (`odata.continue-on-error`)

The `continue-on-error` preference on a batch request is used to request whether, 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 specified with an implicit or explicit value of `true`), or rather stop further processing (if specified with an explicit value of `false`). The syntax of the `continue-on-error` preference is defined in **[OData-ABNF]**.

The `continue-on-error` preference can also be used on a delta update, set-based update, or set-based delete to request that the service continue attempting to process changes after receiving an error.

A service MAY specify support for the `continue-on-error` preference using an annotation with term `Capabilities.BatchContinueOnErrorSupported`, see **[OData-VocCap]**.

The `continue-on-error` preference SHOULD NOT be applied to individual requests within a batch.

Note: The `continue-on-error` preference was named `odata.continue-on-error` in OData version 4.0. Services that support the `continue-on-error` preference SHOULD also support `odata.continue-on-error` for OData 4.0 clients and clients SHOULD use `odata.continue-on-error` for compatibility with OData 4.0 services.

8.2.8.4 Preference `include-annotations` (`odata.include-annotations`)

The `include-annotations` preference in a request for data or metadata is used to specify the set of annotations the client requests to be included, where applicable, in the response.

The value of the `include-annotations` preference is a comma-separated list of namespace-qualified term names or term name patterns to include or exclude, with `*` as a wildcard for name segments. Term names and term name patterns can optionally be followed by a hash (`#`) character and an annotation qualifier. The full syntax of the `include-annotations` preference is defined in **[OData-ABNF]**.

The most specific identifier always takes precedence, with an explicit name taking precedence over a name pattern, and a longer pattern taking precedence over a shorter pattern. 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.

Example 3: a `Prefer` header requesting all annotations within a metadata document to be returned

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

Example 4: a `Prefer` header requesting that no annotations are returned

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

Example 5: a `Prefer` header requesting that all annotations defined under the `display` namespace (recursively) be returned

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

Example 6: a `Prefer` header requesting that the annotation with the term name `subject` within the `display` namespace be returned

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

Example 7: a `Prefer` header requesting that all annotations defined under the `display` namespace (recursively) with the qualifier `tablet` be returned

```
Prefer: include-annotations="display.*#tablet"
```

The `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 `include-annotations` preference.

In the case that the client has specified the `include-annotations` preference in the request, the service SHOULD include a `Preference-Applied` response header containing the `include-annotations` preference to specify the annotations actually included, where applicable, in the response. This value may differ from the annotations requested in the `Prefer` header of the request.

If the `include-annotations` preference is specified on an individual request within a batch, then it specifies the preference for that individual request. Individual requests within a batch that don't include the `include-annotations` preference inherit the preference of the overall batch request.

Note: The `include-annotations` preference was named `odata.include-annotations` in OData version 4.0. Services that support the `include-annotations` preference SHOULD also support `odata.include-annotations` for OData 4.0 clients and clients SHOULD use `odata.include-annotations` for compatibility with OData 4.0 services. If both `include-annotations` and `odata.include-annotations` preferences are specified in the same request, the value of the `include-annotations` preference SHOULD be used.

8.2.8.5 Preference `maxpagesize` (`odata.maxpagesize`)

The `maxpagesize` preference is used to request that each collection within the response contain no more than the number of items specified as the positive integer value of this preference. The syntax of the `maxpagesize` preference is defined in **[OData-ABNF]**.

Example 8: 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 `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 `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 above, 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 entities.

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

The `maxpagesize` preference SHOULD NOT be applied to a batch request, but MAY be applied to individual requests within a batch.

Note: The `maxpagesize` preference was named `odata.maxpagesize` in OData version 4.0. Services that support the `maxpagesize` preference SHOULD also support `odata.maxpagesize` for OData 4.0 clients and clients SHOULD use `odata.maxpagesize` for compatibility with OData 4.0 services. If both `maxpagesize` and `odata.maxpagesize` preferences are specified in the same request, the value of the `maxpagesize` preference SHOULD be used.

8.2.8.6 Preference `omit-values`

The `omit-values` preference specifies values that MAY be omitted from a response payload. Valid values are `nulls` or `defaults`.

If `nulls` is specified, then the service MAY omit properties containing null values from the response, in which case it MUST specify the `Preference-Applied` response header with `omit-values=nulls`.

If `defaults` is specified, then the service MAY omit properties containing default values from the response, including nulls for properties that have no other defined default value. Nulls MUST be included for properties that have a non-null default value defined. If the service omits default values, it MUST specify the `Preference-Applied` response header with `omit-values=defaults`.

Properties with instance annotations are not affected by this preference and MUST be included in the payload if they would be included without this preference. Clients MUST NOT try to reconstruct a null or default value for properties for which an instance annotation is present and no property value is present, for example if the property is omitted due to permissions and has been replaced with the instance annotation `Core.Permissions` and a value of `None`, see **[OData-VocCore]**.

Properties with null or default values MUST be included in delta payloads, if modified.

The response to a `POST` operation MUST include any properties not set to their default value, and the response to a `PUT` or `PATCH` operation MUST include any properties whose values were changed as part of the operation.

The `omit-values` preference does not affect a request payload.

8.2.8.7 Preference `return=representation` and `return=minimal`

The `return=representation` and `return=minimal` preferences are defined in **[RFC7240]**.

In OData, `return=representation` or `return=minimal` is defined for use with a `POST`, `PUT`, or `PATCH` Data Modification Request, or with an Action Request. Specifying a preference of `return=representation` or `return=minimal` in a `GET` or `DELETE` request does not have any effect.

A preference of `return=representation` or `return=minimal` is allowed 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 invoke the request but does not return content in the response. The service MAY apply this preference by returning `204 No Content` in which case 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 resource. The service MAY apply this preference by returning the representation of the successfully modified resource in the body of the response, formatted according to the rules specified for the requested format. In this case the service MAY include a `Preference-Applied` response header containing the `return=representation` preference.

The `return` preference SHOULD NOT be applied to a batch request, but MAY be applied to individual requests within a batch.

8.2.8.8 Preference respond-async

The respond-async preference, as defined in **[RFC7240]**, 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 for batch requests, in which case it applies to the batch request as a whole and not to individual requests within the batch request.

In the case that the service applies the respond-async preference it MUST include a Preference-Applied response header containing the respond-async preference.

A service MAY specify the support for the respond-async preference using an annotation with term Capabilities.AsynchronousRequestsSupported, see **[OData-VocCap]**.

Example 9: 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
```

8.2.8.9 Preference track-changes (odata.track-changes)

The track-changes preference is used to request that the service return a delta link that can subsequently be used to obtain changes (deltas) to this result. The syntax of the track-changes preference is defined in **[OData-ABNF]**.

For paged results, the preference MUST be specified on the initial request. Services MUST ignore the track-changes preference if applied to the next link.

The delta link MUST only be returned on the final page of results in place of the next link.

The service includes a Preference-Applied response header in the first page of the response containing the track-changes preference to signal that changes are being tracked.

A service MAY specify the support for the track-changes preference using an annotation with term Capabilities.ChangeTracking, see **[OData-VocCap]**.

The track-changes preference SHOULD NOT be applied to a batch request, but MAY be applied to individual requests within a batch.

Note: The track-changes preference was named odata.track-changes in OData version 4.0. Services that support the track-changes preference SHOULD also support odata.track-changes for OData 4.0 clients and clients SHOULD use odata.track-changes for compatibility with OData 4.0 services.

8.2.8.10 Preference wait

The wait preference, as defined in **[RFC7240]**, is 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.

If the `wait` preference is specified on an individual request within a batch, then it specifies the maximum amount of time to wait for that individual request. If the `wait` preference is specified on a batch, then it specifies the maximum time to wait for the entire batch.

8.3 Response Headers

In addition to the Common Headers, the following response headers have defined meaning in OData.

8.3.1 Header `AsyncResult`

A 4.01 service MUST include the `AsyncResult` header in `200 OK` responses from a status monitor resource in order to indicate the final HTTP Response Status Code of an asynchronously executed request. The header value is the three-digit HTTP response code, see **[OData-ABNF]**.

The `AsyncResult` header SHOULD NOT be applied to individual responses within a batch.

8.3.2 Header `ETag`

A response MAY include an `ETag` header, see **[RFC9110]**. Services MUST include this header if they require an ETag to be specified when modifying the resource.

Services MUST support specifying the value returned in the `ETag` header in an `If-None-Match` header of a subsequent Data Request for the resource. Clients MUST specify the value returned in the `ETag` header, or star (`*`), in an `If-Match` header of a subsequent Data Modification Request or Action Request in order to apply optimistic concurrency control in updating, deleting, or invoking an action bound to the resource.

As OData allows multiple formats for representing the same structured information, services SHOULD use weak ETags that only depend on the representation-independent entity state. A strong ETag MUST change whenever the representation of an entity changes, so it has to depend on the `Content-Type`, the `Content-Encoding`, and potentially other characteristics of the response.

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.

The `ETag` header SHOULD NOT be included for the overall batch response, but MAY be included in individual responses within a batch.

8.3.3 Header `Location`

The `Location` header MUST be returned in the response from a Create Entity or Create Media Entity request to specify the edit URL, or for read-only entities the read 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.

The `Location` header SHOULD NOT be included for the overall batch response, but MAY be included in individual responses within a batch.

8.3.4 Header `OData-EntityId`

A response to a create or upsert 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. The syntax of the `OData-EntityId` header is defined in **[OData-ABNF]**.

The `OData-EntityID` header SHOULD NOT be included for the overall batch response, but MAY be included in individual responses within a batch.

8.3.5 Header OData-Error

A response with an in-stream error MAY include an `OData-Error` trailing header if the transport protocol supports trailing headers (e.g. HTTP/1.1 with chunked transfer encoding, or HTTP/2).

The value of this trailing header is a standard OData error response according to the OData response format, encoded suitably for transport in a header, see e.g. **[OData-JSON]**.

8.3.6 Header Preference-Applied

In a response to a request that specifies a `Prefer` header, a service MAY include a `Preference-Applied` header, as defined in **[RFC7240]**, 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.

If the `Preference-Applied` header is specified on an individual response within a batch, then it specifies the preferences applied to that individual response. If the `Preference-Applied` header is specified on a batch response, then it specifies the preferences applied to the overall batch.

8.3.7 Header Retry-After

A service MAY include a `Retry-After` header, as defined in **[RFC9110]**), in `202 Accepted` and in `3xx Redirect` responses

The `Retry-After` header specifies the duration of time, in seconds, that the client is asked to wait before retrying the request or issuing a request to the resource returned as the value of the `Location` header.

8.3.8 Header Vary

If a response varies depending on the `OData-Version` of the response, the service MUST include a `Vary` header listing the `OData-MaxVersion` request header field to allow correct caching of the response.

If a response varies depending on the applied preferences (`allow-entityreferences`, `include-annotations`, `omit-values`, `return`), the service MUST include a `Vary` header listing the `Prefer` request header field to allow correct caching of the response.

Alternatively, the server MAY include a `Vary` header with the special value `*` as defined by **[RFC9110]**, Section 8.2.1. Note that this will make it impossible for a proxy to cache the response, see **[RFC7240]**.

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 request that does not create a resource returns `200 OK` if it is completed successfully and the value of the resource is not `null`. 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, or Invoke Action request that successfully creates a resource returns `201 Created`. In this case, the response body MUST contain the resource created.

9.1.3 Response Code 202 Accepted

`202 Accepted` indicates that the Data Service Request has been accepted and has not yet completed executing asynchronously. The asynchronous handling of requests is defined in the sections on Asynchronous Requests and Asynchronous Batch Requests.

9.1.4 Response Code 204 No Content

A request returns `204 No Content` if the requested resource has the `null` value, or if the service applies a `return=minimal` preference. In this case, the response body MUST be empty.

As defined in **[RFC9110]**, a Data Modification Request that responds with `204 No Content` MAY include an `ETag` header with a value reflecting the result of the data modification if and only if the client can reasonably "know" the new representation of the resource without actually receiving it. For a `PUT` request this means that the response body of a corresponding `200 OK` or `201 Created` response would have been identical to the request body, i.e. no server-side modification of values sent in the request body, no server-calculated values etc. For a `PATCH` request this means that the response body of a corresponding `200 OK` or `201 Created` response would have consisted of all values sent in the request body, plus (for values not sent in the request body) server-side values corresponding to the `ETag` value sent in the `If-Match` header of the `PATCH` request, i.e. the previous values "known" to the client.

9.1.5 Response Code 3xx Redirection

As per **[RFC9110]**, a `3xx Redirection` indicates that further action needs to be taken by the client in order to fulfill the request. 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.

9.1.6 Response Code 304 Not Modified

As per **[RFC9110]**, a `304 Not Modified` is returned when the client performs a `GET` request containing an `If-None-Match` header and the content has not changed. In this case the response SHOULD NOT include other headers in order to prevent inconsistencies between cached entity-bodies and updated headers.

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 `304 Not Modified`.

9.2 Client Error Responses

Error codes in the `4xx` range indicate a client error, such as a malformed request.

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

`404 Not Found` indicates that the resource specified by the request URL does not exist. The response body MAY provide additional information.

9.2.2 Response Code 405 Method Not Allowed

`405 Method Not Allowed` indicates that the resource specified by the request URL does not support the request method. In this case the response MUST include an `Allow` header containing a list of valid request methods for the requested resource as defined in **[RFC9110]**.

9.2.3 Response Code 406 Not Acceptable

`406 Not Acceptable` indicates that the resource specified by the request URL does not have a current representation that would be acceptable for the client according to the request headers `Accept`, `Accept-Charset`, and `Accept-Language`, and that the service is unwilling to supply a default representation.

9.2.4 Response Code 410 Gone

`410 Gone` indicates that the requested resource is no longer available. This can happen if a client has waited too long to follow a delta link or a status-monitor-resource link, or a next link on a collection that was requested with snapshot isolation.

9.2.5 Response Code 412 Precondition Failed

As defined in **[RFC9110]**, `412 Precondition Failed` indicates that the client has performed a conditional request and the resource fails the condition. The service MUST ensure that no observable change occurs as a result of the request.

9.2.6 Response Code 424 Failed Dependency

`424 Failed Dependency` indicates that a request was not performed due to a failed dependency; for example, a request within a batch that depended upon a request that failed.

9.3 Server Error Responses

As defined in **[RFC9110]**, error codes in the `5xx` range indicate service errors.

9.3.1 Response Code 501 Not Implemented

If the client requests functionality not implemented by the OData Service, the service MAY respond with `501 Not Implemented` and include a response body describing the functionality not implemented.

9.4 Error Response Body

An error response body can be the result of a failure of OData processing or of the underlying infrastructure. An OData-specific error response (which can be recognized by the presence of the `OData-Version` header) is format-specific and consists at least of the following information:

- `code`: required non-null, non-empty, language-independent string. Its value is a service-defined error code. This code serves as a sub-status for the HTTP error code specified in the response.
- `message`: required non-null, non-empty, language-dependent, human-readable string describing the error. The `Content-Language` header MUST contain the language code from **[RFC5646]** corresponding to the language in which the value for message is written.
- `target`: optional nullable, potentially empty string indicating the target of the error, for example, the name of the property in error.
- `details`: optional, potentially empty collection of instances of structured types with `code`, `message`, and `target` following the rules above.
- `innererror`: optional instance of structured type with service-defined content.

Service implementations SHOULD carefully consider which information to include in production environments to guard against potential security concerns around information disclosure.

9.5 In-Stream Errors

In the case that the service encounters an error after sending a success status to the client, the service MUST leave the response malformed according to its `Content-Type` or abort the response by causing an error on transport protocol level. Clients MUST treat the entire response as being in error.

Services MAY include the header `OData-Error` as a trailing header if supported by the transport protocol (e.g. HTTP/1.1 with chunked transfer encoding, or HTTP/2).

10 Context URL

The *context URL* describes the content of the payload. It consists of the canonical metadata document URL and a fragment identifying the relevant portion of the metadata document. The context URL makes response payloads "self-contained", allowing a recipient to retrieve metadata, resolve references, and construct canonical links omitted from response payloads in certain optimized formats.

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

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

The following subsections describe how the context URL is constructed for each category of payload by providing a *context URL template*. The context URL template uses the following terms:

- `{context-url}` is the canonical resource path to the `$metadata` document,
- `{entity-set}` is the name of an entity set or path to a containment navigation property,
- `{entity}` is the canonical URL for an entity,
- `{singleton}` is the canonical URL for a singleton entity,
- `{select-list}` is an optional parenthesized comma-separated list of selected properties, instance annotations, functions, and actions,
- `{property-path}` is the path to a structural property of the entity,
- `{type-name}` is a qualified type name,
- `{/type-name}` is an optional type-cast segment containing the qualified name of a derived or implemented type prefixed with a forward slash.

The full grammar for the context URL is defined in **[OData-ABNF]**. Note that the syntax of the context URL is independent of whatever URL conventions the service uses for addressing individual entities.

10.1 Service Document

Context URL template:

```
{context-url}
```

The context URL of the service document is the metadata document URL of the service.

Example 10: resource URL and corresponding context URL

```
http://host/service/
http://host/service/$metadata
```

10.2 Collection of Entities

Context URL template:

```
{context-url}#{entity-set}
{context-url}#Collection({type-name})
```

If all entities in the collection are members of one entity set, its name is the context URL fragment.

Example 11: resource URL and corresponding context URL

```
http://host/service/Customers
http://host/service/$metadata#Customers
```

If the entities are contained, then `entity-set` is the top-level entity set or singleton followed by the canonical path to the containment navigation property of the containing entity. Key values in that path are represented in canonical form (parentheses-style) without percent-encoding.

Example 12: resource URL and corresponding context URL for contained entities

```
http://host/service/Orders(4711)/Items
http://host/service/$metadata#Orders(4711)/Items
```

If the entities in the response are not bound to a single entity set, such as from a function or action with no entity set path, a function import or action import with no specified entity set, or a navigation property with no navigation property binding, the context URL fragment specifies the type of the returned entity collection.

10.3 Entity

Context URL template:

```
{context-url}#{entity-set}/$entity
{context-url}#{type-name}
```

If a response or response part is a single entity of the declared type of an entity set, the context URL fragment is the entity set's name with `/$entity` appended.

Example 13: resource URL and corresponding context URL

```
http://host/service/Customers(1)
http://host/service/$metadata#Customers/$entity
```

If the entity is contained, then `entity-set` is the top-level entity set or singleton followed by the path to the containment navigation property of the containing entity.

Example 14: resource URL and corresponding context URL for contained entity

```
http://host/service/Orders(4711)/Items(1)
http://host/service/$metadata#Orders(4711)/Items/$entity
```

If the entity is not bound to an entity set, such as an entity returned from a function or action with no entity set path, a function import or action import with no specified entity set, or a navigation property with no navigation property binding, the context URL fragment specifies the type of the returned entity.

10.4 Singleton

Context URL template:

```
{context-url}#{singleton}
```

If a response or response part is a singleton, its name is the context URL fragment.

Example 15: resource URL and corresponding context URL

```
http://host/service/MainSupplier
http://host/service/$metadata#MainSupplier
```

10.5 Collection of Derived Entities

Context URL template:

```
{context-url}#{entity-set}{/type-name}
```

If an entity set consists exclusively of derived entities, a type-cast segment is added to the context URL.

Example 16: resource URL and corresponding context URL

```
http://host/service/Customers/Model.VipCustomer
http://host/service/$metadata#Customers/Model.VipCustomer
```

10.6 Derived Entity

Context URL template:

```
{context-url}#{entity-set}{/type-name}/$entity
```

If a response or response part is a single entity of a type derived from the declared type of an entity set, a type-cast segment is appended to the entity set name.

Example 17: resource URL and corresponding context URL

```
http://host/service/Customers(2)/Model.VipCustomer
http://host/service/$metadata#Customers/Model.VipCustomer/$entity
```

10.7 Collection of Projected Entities

Context URL templates:

```
{context-url}#{entity-set}{/type-name}{select-list}
{context-url}#Collection({type-name}){select-list}
```

If a result contains only a subset of properties, the parenthesized comma-separated list of the selected defined or dynamic properties, instance annotations, navigation properties, functions, and actions is appended to the context URL representing the collection of entities.

Regardless of how contained structural properties are represented in the request URL (as paths or as select options) they are represented in the context URL using path syntax, as defined in OData 4.0.

The shortcut `*` represents the list of all structural properties. Properties defined on types derived from the declared type of the entity set (or type specified in the type-cast segment if specified) are prefixed with the qualified name of the derived type as defined in **[OData-ABNF]**.

The list also contains explicitly selected or expanded instance annotations. It is possible to select or expand only instance annotations, in which case only those selected or expanded annotations appear in the result. Note that annotations specified only in the `include-annotations` preference do not appear in the context URL and do not affect the selected/expanded properties.

Operations in the context URL are represented using the namespace- or alias-qualified name. Function names suffixed with parentheses represent a specific overload, while function names without parentheses represent all overloads of the function.

OData 4.01 responses MAY use the shortcut pattern `{namespace}.*` to represent the list of all bound actions or functions available for entities in the collection, see system query option `$select`.

Example 18: resource URL and corresponding context URL

```
http://host/service/Customers?$select=Address,Orders,Model.VipCustomer/PreferredContact
http://host/service/$metadata#Customers(Address,Orders,Model.VipCustomer/PreferredContact)
```

10.8 Projected Entity

Context URL templates:

```
{context-url}#{entity-set}{/type-name}{select-list}/$entity
{context-url}#{singleton}{select-list}
{context-url}#{type-name}{select-list}
```

If a single entity contains a subset of properties, the parenthesized comma-separated list of the selected defined or dynamic properties, instance annotations, navigation properties, functions, and actions is appended to the `{entity-set}` after an optional type-cast segment and prior to appending `/$entity`. If the response is not a subset of a single entity set, the `{select-list}` is instead appended to the `{type-name}` of the returned entity.

Regardless of how contained structural properties are represented in the request URL (as paths or as select options) they are represented in the context URL using path syntax, as defined in OData 4.0.

The shortcut `*` represents the list of all structural properties. Properties defined on types derived from the type of the entity set (or type specified in the type-cast segment if specified) are prefixed with the qualified name of the derived type as defined in **[OData-ABNF]**. Note that expanded properties are automatically included in the response.

The list also contains explicitly selected or expanded instance annotations. It is possible to select or expand only instance annotations, in which case only those selected or expanded annotations appear in the result. Note that annotations specified only in the `include-annotations` preference do not appear in the context URL and do not affect the selected/expanded properties.

Operations in the context URL are represented using the namespace- or alias-qualified name. Function names suffixed with parentheses represent a specific overload, while function names without parentheses represent all overloads of the function.

OData 4.01 responses MAY use the shortcut pattern `{namespace}.*` to represent the list of all bound actions or functions available for the returned entity, see system query option `$select`.

Example 19: resource URL and corresponding context URL

```
http://host/service/Customers(1)?$select=Name,Rating
http://host/service/$metadata#Customers(Name,Rating)/$entity
```

10.9 Collection of Expanded Entities

Context URL template:

```
{context-url}#{entity-set}{/type-name}{select-list}
{context-url}#Collection({type-name}){select-list}
```

For a 4.01 response, if a navigation property is explicitly expanded, then in addition to any non-suffixed names of any selected properties, navigation properties, functions or actions, the comma-separated list of properties MUST include the name of the expanded property, suffixed with the parenthesized comma-separated list of any properties of the expanded navigation property that are selected or expanded. If the expanded navigation property does not contain a nested `$select` or `$expand`, then the expanded property is suffixed with empty parentheses. If the expansion is recursive for nested children, a plus sign (+) is infixed between the navigation property name and the opening parenthesis.

For a 4.0 response, the expanded navigation property suffixed with parentheses is omitted from the select-list if it does not contain a nested `$select` or `$expand`, but MUST still be present, without a suffix, if it is explicitly selected.

The context URL has no shortcut for representing the list of all navigation properties; `$expand=*` is treated as if all navigation properties were explicitly expanded.

If the context URL includes only expanded navigation properties (i.e., only navigation properties suffixed with parentheses), then all structural properties are implicitly selected (same as if there were no properties listed in the select-list).

Navigation properties with expanded references are not represented in the context URL.

Example 20: resource URL and corresponding context URL — select and expand

```
http://host/service/Customers?$select=Name&$expand=Address/Country
http://host/service/$metadata#Customers(Name,Address/Country())
```

Example 21: resource URL and corresponding context URL — expand `$ref`

```
http://host/service/Customers?$expand=Orders/$ref
http://host/service/$metadata#Customers
```

Example 22: resource URL and corresponding context URL — expand with `$levels`

```
http://host/service/Employees/Sales.Manager?$select=DirectReports
        &$expand=DirectReports($select=FirstName,LastName;$levels=4)
http://host/service/$metadata
        #Employees/Sales.Manager(DirectReports,DirectReports+(FirstName,LastName))
```

10.10 Expanded Entity

Context URL template:

```
{context-url}#{entity-set}{/type-name}{select-list}/$entity
{context-url}#{singleton}{select-list}
{context-url}#{type-name}{select-list}
```

For a 4.01 response, if a navigation property is explicitly expanded, then in addition to the non-suffixed names of any selected properties, navigation properties, functions or actions, the comma-separated list of properties MUST include the name of the expanded property, suffixed with the parenthesized comma-separated list of any properties of the expanded navigation property that are selected or expanded. If the expanded navigation property does not contain a nested `$select` or `$expand`, then the expanded property is suffixed with empty parentheses. If the expansion is recursive for nested children, a plus sign (+) is infixed between the navigation property name and the opening parenthesis.

For a 4.0 response, the expanded navigation property suffixed with parentheses is omitted from the select-list if it does not contain a nested `$select` or `$expand`, but MUST still be present, without a suffix, if it is explicitly selected.

If the context URL includes only expanded navigation properties (i.e., only navigation properties suffixed with parentheses), then all structural properties are implicitly selected (same as if there were no properties listed in the select-list).

Navigation properties with expanded references are not represented in the context URL.

Example 23: resource URL and corresponding context URL

```
http://host/service/Employees(1)/Sales.Manager?
        $expand=DirectReports($select=FirstName,LastName;$levels=4)
http://host/service/$metadata
        #Employees/Sales.Manager(DirectReports+(FirstName,LastName))/$entity
```

10.11 Collection of Entity References

Context URL template:

```
{context-url}#Collection($ref)
```

If a response is a collection of entity references, the context URL does not contain the type of the referenced entities.

Example 24: resource URL and corresponding context URL for a collection of entity references

```
http://host/service/Customers('ALFKI')/Orders/$ref
http://host/service/$metadata#Collection($ref)
```

10.12 Entity Reference

Context URL template:

```
{context-url}#$ref
```

If a response is a single entity reference, **`$ref`** is the context URL fragment.

Example 25: resource URL and corresponding context URL for a single entity reference

```
http://host/service/Orders(10643)/Customer/$ref
http://host/service/$metadata#$ref
```

10.13 Property Value

Context URL templates:

```
{context-url}#{entity}/{property-path}{select-list}
{context-url}#{type-name}{select-list}
```

If a response represents an individual property of an entity with a canonical URL, the context URL specifies the canonical URL of the entity and the path to the structural property of that entity. The path MUST include cast segments for properties defined on types derived from the expected type of the previous segment.

If the property value does not contain explicitly or implicitly selected navigation properties or operations, OData 4.01 responses MAY use the less specific second template.

Example 26: resource URL and corresponding context URL

```
http://host/service/Customers(1)/Addresses
http://host/service/$metadata#Customers(1)/Addresses
```

10.14 Collection of Complex or Primitive Types

Context URL template:

```
{context-url}#Collection({type-name}){select-list}
```

If a response is a collection of complex types or primitive types that do not represent an individual property of an entity with a canonical URL, the context URL specifies the fully qualified type of the collection.

Example 27: resource URL and corresponding context URL

```
http://host/service/TopFiveHobbies()
http://host/service/$metadata#Collection(Edm.String)
```

10.15 Complex or Primitive Type

Context URL template:

```
{context-url}#{type-name}{select-list}
```

If a response is a complex type or primitive type that does not represent an individual property of an entity with a canonical URL, the context URL specifies the fully qualified type of the result.

Example 28: resource URL and corresponding context URL

```
http://host/service/MostPopularName()
http://host/service/$metadata#Edm.String
```

10.16 Operation Result

Context URL templates:

```
{context-url}#{entity-set}{/type-name}{select-list}
{context-url}#{entity-set}{/type-name}{select-list}/$entity
{context-url}#{entity}/{property-path}{select-list}
{context-url}#Collection({type-name}){select-list}
{context-url}#{type-name}{select-list}
```

If the response from an action or function is a collection of entities or a single entity that is a member of an entity set, the context URL identifies the entity set. If the response from an action or function is a property of a single entity, the context URL identifies the entity and property. Otherwise, the context URL identifies the type returned by the operation. The context URL will correspond to one of the former examples.

Example 29: resource URL and corresponding context URL

```
http://host/service/TopFiveCustomers()
http://host/service/$metadata#Customers
```

10.17 Delta Payload Response

Context URL template:

```
{context-url}#{entity-set}{/type-name}{select-list}/$delta
{context-url}#{entity}{select-list}/$delta
{context-url}#{entity}/{property-path}{select-list}/$delta
#$delta
```

The context URL of a delta response is the context URL of the response to the defining query, followed by `/$delta`. This includes singletons, single-valued navigation properties, and collection-valued navigation properties.

If the entities are contained, then `{entity-set}` is the top-level entity set followed by the path to the containment navigation property of the containing entity.

Example 30: resource URL and corresponding context URL

```
http://host/service/Customers?$deltatoken=1234
http://host/service/$metadata#Customers/$delta
```

The context URL of an update request body for a collection of entities is simply the fragment `#$delta`.

10.18 Item in a Delta Payload Response

Context URL templates:

```
{context-url}#{entity-set}/$deletedEntity
{context-url}#{entity-set}/$link
{context-url}#{entity-set}/$deletedLink
```

In addition to new or changed entities which have the canonical context URL for an entity, a delta response can contain deleted entities, new links, and deleted links. They are identified by the corresponding context URL fragment. `{entity-set}` corresponds to the set of the deleted entity, or source entity for an added or deleted link.

10.19 `$all` Response

Context URL template:

```
{context-url}#Collection(Edm.EntityType)
```

Responses to requests to the virtual collection `$all` (see **[OData-URL]**) use the built-in abstract entity type. Each single entity in such a response has its individual context URL that identifies the entity set or singleton.

10.20 `$crossjoin` Response

Context URL template:

```
{context-url}#Collection(Edm.ComplexType)
```

Responses to requests to the virtual collections `$crossjoin(...)` (see **[OData-URL]**) use the built-in abstract complex type. Single instances in these responses do not have a context URL.

11 Data Service Requests

This chapter describes the semantics of the HTTP verbs `GET`, `POST`, `PATCH`, `PUT`, and `DELETE` for OData resources.

`GET` requests:

- Metadata Requests and subsections
- Requesting Data and subsections
- Requesting Changes and subsections
- Functions and subsections

`POST` requests:

- Create an Entity and subsections
- Create a Media Entity
- Positional Inserts
- Actions and subsections
- Batch Requests and subsections

`PATCH` and `PUT` requests:

- Update an Entity and subsections
- Upsert an Entity
- Modifying Relationships between Entities and subsections
- Update a Media Entity Stream
- Update Stream Values
- Update a Primitive Property
- Update a Complex Property
- Update a Collection Property
- Managing Members of an Ordered Collection
- Update a Collection of Entities
- Update Members of a Collection

`DELETE` requests:

- Delete an Entity
- Delete a Media Entity
- Delete Stream Values
- Set a Value to Null
- Delete Members of a Collection

11.1 Metadata Requests

An OData service is a self-describing service that exposes metadata defining the entity sets, singletons, 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.

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-CSDLJSON] describes a JSON representation for OData metadata documents and provides a JSON schema to validate their contents. The media type of the JSON representation of an OData metadata document is `application/json`.

[OData-CSDLXML] 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 can expose a metadata document that describes the data model exposed by the service. The *metadata document URL* MUST 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.

11.2 Requesting Data

OData services support requests for data via HTTP `GET` requests.

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

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

OData services are hypermedia driven services that return URLs to the client. If a client subsequently requests the advertised resource and the URL has expired, then the service SHOULD respond with `410 Gone`. If this is not feasible, the service MUST respond with `404 Not Found`.

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 `$format` query option. If the client specifies neither an `Accept` header nor the `$format` query option, the service is allowed to return the response in any format.

11.2.1 System Query Options

OData defines a number of system query options that allow refining the request. System query options are prefixed with the dollar (`$`) character, which is optional in OData 4.01. 4.01 services MUST support case-insensitive system query option names specified with or without the `$` prefix. Clients that want to work with 4.0 services MUST use lower case names and specify the `$` prefix.

The result of the request MUST be as if the system query options were evaluated in the following order.

- `$schemaversion` MUST be evaluated first, because it may influence any further processing.

Prior to applying any server-driven paging:

- `$apply` — defined in **[OData-Aggregation]**
- `$compute`
- `$search`
- `$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 a URL that identifies the entity, e.g. its read URL.

The read URL can be obtained from a response payload containing that instance, for example as a `readLink` or `editLink` in an **[OData-JSON]** payload. In addition, services MAY support conventions for constructing a read URL using the entity's key value(s), as described in **[OData-URL]**.

The set of structural or navigation properties to return may be specified through `$select` or `$expand` system query options.

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.

Properties that are not available are not returned. If their unavailability is due to permissions, the `Core.Permissions` annotation, defined in **[OData-VocCore]** MUST be returned for the property with a value of `None`. If the `omit-values` preference is applied, `Core.Permissions` or another specific annotation that explains the reason MUST be returned for every unavailable property.

If no entity exists with the specified request URL, the service responds with `404 Not Found`.

11.2.3 Requesting the Media Stream of a Media Entity using $value

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

Use a media entity if the out-of-band stream is the main topic of interest and the media entity is just structured additional information attached to the stream. Use a normal entity with one or more stream properties if the structured data of the entity is the main topic of interest and the stream data is just additional information attached to the structured data.

To address the media stream represented by a media entity, clients append `/$value` to the resource path of the media entity URL. The media type of the response is the media type of the stream, subject to content type negotiation based on the `Accept` header of the request. The response body is the octet-stream that represents the raw value of the media stream with that media type. Alternatively, services MAY redirect from this canonical URL to the source URL of the media stream.

Appending `/$value` to an entity that is not a media entity returns `400 Bad Request`.

Attempting to retrieve the media stream from a single-valued navigation property referencing a media entity whose value is null returns `404 Not Found`.

11.2.4 Requesting Individual Properties

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

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

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

If the property is single-valued and has the `null` value, the service responds with `204 No Content`.

If the property is not available, for example due to permissions, the service responds with `404 Not Found`.

Example 31:

```
GET http://host/service/Products(1)/Name
```

11.2.4.1 Requesting Stream Properties

If the property being requested has type `Edm.Stream` (see **[OData-URL, section 9]**), the media type of the response is the media type of the stream, subject to content type negotiation based on the `Accept` header of the request. The response body is the octet-stream that represents the raw value of the stream property with that media type.

Note this response format disregards any `$format` system query option.

11.2.4.2 Requesting a Raw Value using `$value`

To retrieve the raw value of a primitive property or operation result, the client sends a `GET` request to the raw value URL. See the **[OData-URL]** document for details.

The `Content-Type` of the response is determined using the `Accept` header and the `$format` system query option.

The default format for `Edm.Binary` is the format specified by the `Core.MediaType` annotation (see **[OData-VocCore]**) if this annotation is present. If not annotated, the format cannot be predicted by the client.

The default format for `Edm.Geo` types is `text/plain` using the WKT (well-known text) format, see rules `fullCollectionLiteral`, `fullLineStringLiteral`, `fullMultiPointLiteral`, `fullMultiLineStringLiteral`, `fullMultiPolygonLiteral`, `fullPointLiteral`, and `fullPolygonLiteral` in **[OData-ABNF]**.

The default format for single primitive values except `Edm.Binary` and the `Edm.Geo` types is `text/plain`. Responses of type `Edm.String` can use the `charset` format parameter to specify the character set used for representing the string value. Responses for the other primitive types follow the rules `booleanValue`, `byteValue`, `dateValue`, `dateTimeOffsetValue`, `decimalValue`, `doubleValue`, `durationValue`, `enumValue`, `guidValue`, `int16Value`, `int32Value`, `int64Value`, `sbyteValue`, `singleValue`, and `timeOfDayValue` in **[OData-ABNF]**.

A raw value request for a property or operation result of type `Edm.Stream` returns `400 Bad Request`.

A raw value request for a property or operation result that is `null` results in a `204 No Content` response.

If the property or operation result is not available, for example due to permissions, the service responds with `404 Not Found`.

Example 32:

```
GET http://host/service/Products(1)/Name/$value
```

11.2.5 Specifying Properties to Return

The `$select` and `$expand` system query options enable the client to specify the set of structural properties and navigation properties to include in a response. The service MAY include additional properties not specified in `$select` and `$expand`, including properties not defined in the metadata document.

11.2.5.1 System Query Option `$select`

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 returns the specified content, if available, along with any available expanded navigation or stream properties, and MAY return additional information.

The value of the `$select` query option is a comma-separated list of paths that end with properties, non-entity-valued instance annotations, qualified action names, or qualified function names, as well as of the star operator (`*`), or the star operator prefixed with the namespace or alias of the schema in order to specify all operations defined in the schema. Only aliases defined in the metadata document of the service can be used in URLs.

Example 33: request only the `Rating` *and* `ReleaseDate` *for the matching Products*

```
GET http://host/service/Products?$select=Rating,ReleaseDate
```

It is also possible to request all structural properties, including any dynamic properties, using the star operator. The star operator SHOULD NOT introduce navigation properties, actions or functions not otherwise requested.

Example 34:

```
GET http://host/service/Products?$select=*
```

Properties of related entities can be specified by including the `$select` query option within the `$expand`.

Example 35:

```
GET http://host/service/Products?$expand=Category($select=Name)
```

The properties specified in `$select` are represented in addition to any expanded navigation or stream properties. If a navigation property is specified in `$select`, then the corresponding navigation link is represented in the response. If the navigation property also appears in an `$expand` query option, then it is additionally represented as inline content.

Example 36: for each category, return the `CategoryName` *and the* `Products` *navigation link*

```
GET http://host/service/Categories?$select=CategoryName,Products
```

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

Example 37:

```
GET http://host/service/Products?$select=DemoService.*
```

Query options can be applied to a selected property by appending a semicolon-separated list of query options, enclosed in parentheses, to the property. Allowed system query options are `$select` and `$compute` for complex properties, plus `$filter`, `$search`, `$count`, `$orderby`, `$skip`, and `$top` for collection-valued properties. A property MUST NOT have select options specified in more than one place in a request and MUST NOT have both select options and expand options specified.

If the `$select` query option is not specified, [the service returns the full set of properties or a default set of properties. The default set of properties MUST include all key properties. Services may change the default set of properties returned. This includes returning new properties by default and omitting properties previously returned by default. Clients that rely on specific properties in the response MUST use `$select` with the required properties or with `*`.

If the service returns less than the full set of properties, either because the client specified a select or because the service returned a subset of properties in the absence of a select, the context URL MUST reflect the set of selected properties and projected expanded navigation properties.

11.2.5.2 System Query Option `$expand`

The `$expand` system query option indicates the related entities and stream values that MUST be represented inline. The service MUST return the specified content, and MAY choose to return additional information.

The value of `$expand` is a comma-separated list of expand items. Each expand item is evaluated relative to the retrieved resource being expanded.

For a full description of the syntax used when building requests, see **[OData-URL]**, section 5.1.3.

Example 38: for each customer entity within the Customers entity set the value of all related Orders will be represented inline

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

Example 39: for each customer entity within the Customers entity set the references to the related Orders will be represented inline

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

Example 40: for each customer entity within the Customers entity set the media stream representing the customer photo will be represented inline

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

11.2.5.2.1 Expand Options

The set of expanded entities can be further refined through the application of expand options, expressed as a semicolon-separated list of system query options, enclosed in parentheses, see **[OData-URL]**.

Allowed system query options are `$compute`, `$select`, `$expand`, and `$levels` for all navigation properties, plus `$filter`, `$orderby`, `$skip`, `$top`, `$count`, and `$search` for collection-valued navigation properties.

Example 41: for each customer entity within the `Customers` entity set, the value of those related `Orders` whose `Amount` is greater than 100 will be represented inline

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

Example 42: for each order within the `Orders` entity set, the following will be represented inline:

- *The `Items` related to the `Orders` identified by the resource path section of the URL and the products related to each order item.*
- *The `Customer` related to each order returned.*

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

Example 43: for each customer entity in the Customers entity set, the value of all related InHouseStaff will 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 may be returned with no inline representation for the expanded navigation property `InHouseStaff` (the service can always send more than requested)

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

11.2.5.2.1.1 Expand Option `$levels`

The `$levels` expand option can be used to specify the number of levels of recursion for a hierarchy in which the related entity type is the same as, or can be cast to, the source entity type. A `$levels` option with a value of 1 specifies a single expand with no recursion. All provided expand options except `$levels` are applied at each level of the hierarchy.

Services MAY support the symbolic value `max` in addition to numeric values. In that case they MUST solve circular dependencies by injecting an entity reference somewhere in the circular dependency.

Clients using `$levels=max` MUST be prepared to handle entity references in cases were a circular reference would occur otherwise.

4.01 services that support `max` SHOULD do so in a case-insensitive manner. Clients that want to work with 4.0 services MUST use lower case.

Example 44: return each employee from the Employees entity set and, for each employee that is a manager, return all direct reports, recursively to four levels

```
GET http://host/service/Employees?$expand=Model.Manager/DirectReports($levels=4)
```

11.2.5.3 System Query Option `$compute`

The `$compute` system query option allows clients to define computed properties that can be used in a `$select` or within a `$filter` or `$orderby` expression.

Computed properties SHOULD be included as dynamic properties in the result and MUST be included if `$select` is specified with the computed property name, or star (`*`).

Example 45: compute total price for order items (line breaks only for readability)

```
GET http://host/service/Customers
   ?$filter=Orders/any(o:o/TotalPrice gt 100)
   &$expand=Orders($compute=Price mult Qty as TotalPrice
                  ;$select=Name,Price,Qty,TotalPrice)
```

11.2.6 Querying Collections

OData services support querying collections of entities, complex type instances, and primitive values.

The target collection is specified through a URL, and query operations such as filter, sort, paging, and projection are specified as *system query options* optionally prefixed with a dollar (`$`) character. 4.01 Services MUST support case-insensitive system query option names specified with or without the `$` prefix. Clients that want to work with 4.0 services MUST use lower case names and specify the `$` prefix.

The same system query option MUST NOT be specified more than once for any resource.

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`.

11.2.6.1 System Query Option `$filter`

The `$filter` system query option restricts the set of items returned.

Example 46: return all Products whose `Price` is less than $10.00

```
GET http://host/service/Products?$filter=Price lt 10.00
```

The `$count` segment may be used within a `$filter` expression to limit the items returned based on the exact count of related entities or items within a collection-valued property.

Example 47: return all Categories with less than 10 products

```
GET http://host/service/Categories?$filter=Products/$count lt 10
```

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

11.2.6.1.1 Built-in Filter Operations

OData supports a set of built-in filter operations, as described in this section.

4.01 services MUST support case-insensitive operation names. Clients that want to work with 4.0 services MUST use lower case operation names.

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

Operator	Description	Example
Comparison Operators		
`eq`	Equal	`Address/City eq 'Redmond'`
`ne`	Not equal	`Address/City ne 'London'`
`gt`	Greater than	`Price gt 20`
`ge`	Greater than or equal	`Price ge 10`
`lt`	Less than	`Price lt 20`
`le`	Less than or equal	`Price le 100`
`has`	Has flags	`Style has Sales.Color'Yellow'`
`in`	Is a member of	`Address/City in ('Redmond', 'London')`
Logical Operators		
`and`	Logical and	`Price le 200 and Price gt 3.5`
`or`	Logical or	`Price le 3.5 or Price gt 200`
`not`	Logical negation	`not endswith(Description,'milk')`
Arithmetic Operators		
`add`	Addition	`Price add 5 gt 10`
`sub`	Subtraction	`Price sub 5 gt 10`
`mul`	Multiplication	`Price mul 2 gt 2000`
`div`	Division	`Price div 2 gt 4`
`divby`	Decimal Division	`Price divby 2 gt 3.5`
`mod`	Modulo	`Price mod 2 eq 0`
Grouping Operators		
`()`	Precedence grouping	`(Price sub 5) gt 10`

11.2.6.1.2 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.

4.01 services MUST support case-insensitive built-in function names. Clients that want to work with 4.0 services MUST use lower case names.

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 and Collection Functions	
`concat`	`concat(concat(City,', '), Country) eq 'Berlin, Germany'`
`contains`	`contains(CompanyName,'freds')`
`endswith`	`endswith(CompanyName,'Futterkiste')`
`indexof`	`indexof(CompanyName,'lfreds') eq 1`
`length`	`length(CompanyName) eq 19`
`startswith`	`startswith(CompanyName,'Alfr')`
`substring`	`substring(CompanyName,1) eq 'lfreds Futterkiste'`
Collection Functions	
`hassubset`	`hassubset([4,1,3],[3,1])`
`hassubsequence`	`hassubsequence([4,1,3,1],[1,1])`
String Functions	
`matchespattern`	`matchespattern(CompanyName,'%5EA.*e$')`
`tolower`	`tolower(CompanyName) eq 'alfreds futterkiste'`
`toupper`	`toupper(CompanyName) eq 'ALFREDS FUTTERKISTE'`
`trim`	`trim(CompanyName) eq 'Alfreds Futterkiste'`
Date and Time Functions	
`day`	`day(StartTime) eq 8`
`date`	`date(StartTime) ne date(EndTime)`
`fractionalseconds`	`second(StartTime) eq 0`
`hour`	`hour(StartTime) eq 1`
`maxdatetime`	`EndTime eq maxdatetime()`
`mindatetime`	`StartTime eq mindatetime()`
`minute`	`minute(StartTime) eq 0`
`month`	`month(BirthDate) eq 12`
`now`	`StartTime ge now()`

second	`second(StartTime) eq 0`
time	`time(StartTime) le StartOfDay`
totaloffsetminutes	`totaloffsetminutes(StartTime) eq 60`
totalseconds	`totalseconds(duration'PT1M') eq 60`
year	`year(BirthDate) eq 0`
Arithmetic Functions	
ceiling	`ceiling(Freight) eq 33`
floor	`floor(Freight) eq 32`
round	`round(Freight) eq 32`
Type Functions	
cast	`cast(ShipCountry,Edm.String)`
isof	`isof(NorthwindModel.Order)`
isof	`isof(ShipCountry,Edm.String)`
Geo Functions	
geo.distance	`geo.distance(CurrentPosition,TargetPosition)`
geo.intersects	`geo.intersects(Position,TargetArea)`
geo.length	`geo.length(DirectRoute)`
Conditional Functions	
case	`case(X gt 0:1,X lt 0:-1,true:0)`

11.2.6.1.3 Parameter Aliases

Parameter aliases can be used in place of literal values in entity keys, function parameters, or within a `$compute`, `$filter` or `$orderby` expression. 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.

Example 48: returns all employees whose Region property matches the string parameter value `WA`

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

Parameter aliases allow the same value to be used multiple times in a request and may be used to reference primitive, structured, 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. A parameter alias can be used in multiple places within a request URL, but its value MUST NOT be specified more than once.

Parameter alias values used in `/$filter` path segments are always passed as expressions (because that is the expected type of the parameter).

All other parameter alias values are evaluated in the context of the resource identified by the path segment in which they are assigned and passed as values into the expression. Parameter alias value assignments MAY be nested within `$expand` and `$select`, in which case they are evaluated relative to the resource context of the `$expand` or `$select`.

Example 49: returns all employees, expands their manager, and expands all direct reports with the same first name as the manager, using a parameter alias for `$this` to pass the manager into the filter on the expanded direct reports

```
GET http://host/service.svc/Employees?
$expand=Manager(@m=$this;$expand=DirectReports($filter=@m/FirstName eq FirstName))
```

11.2.6.2 System Query Option `$orderby`

The `$orderby` System Query option specifies the order in which items 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 items. 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. Only aliases defined in the metadata document of the service can be used in URLs.

The expression can 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. 4.01 services MUST support case-insensitive values for `asc` and `desc`. Clients that want to work with 4.0 services MUST use lower case values.

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

Items are sorted by the result values of the first expression, and then items with the same value for the first expression are sorted by the result value of the second expression, and so on.

The Boolean value false comes before the value true in ascending order.

Services SHOULD order language-dependent strings according to the `Content-Language` of the response, and SHOULD annotate string properties with language-dependent order with the term `Core.IsLanguageDependent`, see **[OData-VocCore]**.

Values of type `Edm.Stream` or any of the `Geo` types cannot be sorted.

Example 50: return all Products ordered by release date in ascending order, then by rating in descending order

```
GET http://host/service/Products?$orderby=ReleaseDate asc, Rating desc
```

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

Example 51: return all Categories, and their Products ordered according to release date and in descending order of rating

```
GET http://host/service/Categories?$expand=Products($orderby=ReleaseDate asc, Rating desc)
```

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

Example 52: return all Categories ordered by the number of Products within each category

```
GET http://host/service/Categories?$orderby=Products/$count
```

11.2.6.3 System Query Option `$top`

The `$top` system query option specifies a non-negative integer n that limits the number of items returned from a collection.

Let A be a copy of the result set with a total order that extends any existing order of the result set but is otherwise chosen by the service. If no unique ordering is imposed through an `$orderby` query option, the service MUST choose a stable ordering across requests that include `$top` or `$skip`.

If A contains more than n instances, the result of `$top` $= n$ consists of the first n instances in A. Otherwise, the result equals A. The instances in the result are in the same order as they occur in A.

Example 53: return only the first five products of the Products entity set

```
GET http://host/service/Products?$top=5
```

11.2.6.4 System Query Option `$skip`

The `$skip` system query option specifies a non-negative integer n that excludes the first n items of the queried collection from the result.

Let A be a copy of the result set with a total order that extends any existing order of the result set but is otherwise chosen by the service. If no unique ordering is imposed through an `$orderby` query option, the service MUST choose a stable ordering across requests that include `$top` or `$skip`.

If A contains n or fewer instances, the result of `$skip` $= n$ is empty. Otherwise, the first n instances in A are omitted from the result and all remaining instances are kept in the same order as they occur in A.

Example 54: return products starting with the 6th product of the `Products` *entity set*

```
GET http://host/service/Products?$skip=5
```

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

Example 55: return the third through seventh products of the `Products` *entity set*

```
GET http://host/service/Products?$top=5&$skip=2
```

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

11.2.6.5 System Query Option `$count`

The `$count` system query option with a value of `true` specifies that the total count of items within a collection matching the request be returned along with the result.

Example 56: return, along with the results, the total number of products in the collection

```
GET http://host/service/Products?$count=true
```

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

Example 57:

```
GET http://host/service/Categories?$expand=Products($count=true)
```

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

The service returns an HTTP Status code of 400 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 `$filter` and `$search`. Clients should be aware that the count returned inline may not exactly equal the actual number of items returned, due to latency between calculating the count and enumerating the last value or due to inexact calculations on the service.

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

11.2.6.6 System Query Option `$search`

The `$search` system query option restricts the result to include only those items *matching* the specified search expression. The definition of what it means to match is dependent upon the implementation.

Example 58: return all Products that match the search term `bike`

```
GET http://host/service/Products?$search=bike
```

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

Example 59: return all Products that match the phrase `mountain bike`

```
GET http://host/service/Products?$search="mountain bike"
```

The upper-case keyword `NOT` restricts the set of entities to those that do not match the specified term.

Example 60: return all Products that do not match `clothing`

```
GET http://host/service/Products?$search=NOT clothing
```

Multiple terms within a search expression are separated by a space (implicit `AND`) or the upper-case keyword `AND`, indicating that all such terms must be matched.

Example 61: return all Products that match both `mountain` *and* `bike`

```
GET http://host/service/Products?$search=mountain AND bike
```

The upper-case keyword `OR` is used to return entities that satisfy either the immediately preceding or subsequent expression.

Example 62: return all Products that match `mountain` *or* `bike`

```
GET http://host/service/Products?$search=mountain OR bike
```

Parentheses within the search expression group together multiple expressions.

Example 63: return all Products that match `mountain` *or* `bike` *and do not match clothing*

```
GET http://host/service/Products?$search=(mountain OR bike) AND NOT 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 items satisfying both criteria are returned.

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

11.2.6.7 Server-Driven Paging

Responses that include only a partial set of the items identified by the request URL MUST contain a link that allows retrieving the next partial set of items. This link is called a *next link*; its representation is format-specific. The final partial set of items MUST NOT contain a next link.

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

OData clients MUST treat the URL of the next link as opaque, and MUST NOT append system query options to the URL of a next link. Services may not allow a change of format on requests for subsequent pages using the next link. Clients therefore SHOULD request the same format on subsequent page requests using a compatible `Accept` header. 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.

11.2.6.8 Requesting an Individual Member of an Ordered Collection

Individual members of collections of primitive and complex types annotated with the `Ordered` term (see **[OData-VocCore]**) are addressable by appending a segment containing the zero-based ordinal to the URL of the collection. A negative ordinal indexes from the end of the collection, with -1 representing the last item in the collection.

Entities are stably addressable using their canonical URL and are not accessible using an ordinal index.

Example 64: the first address in a list of addresses for `MainSupplier`

```
GET http://host/service/MainSupplier/Addresses/0
```

11.2.7 Requesting Related Entities

To request related entities according to a particular relationship, the client issues a `GET` request to 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 returns `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 service returns `204 No Content`.

Example 65: return the supplier of the product with `ID=1` *in the Products entity set*

```
GET http://host/service/Products(1)/Supplier
```

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 identify an entity or a collection of entities, the service returns `404 Not Found`.

If the resource path identifies 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. The response MAY contain an `ETag` header for the collection whose value changes if the collection of references changes, i.e. a reference is added or removed.

If the resource path identifies a single existing entity, the response MUST be the format-specific representation of an entity reference. The response MAY contain an ETag header which represents the identity of the referenced entity. If the resource path terminates in a single-valued navigation path, the ETag value changes if the relationship is changed and points to a different OData entity. If the resource path is the canonical path for a single entity, the returned ETag can never change.

If the resource path terminates on a single entity and no such entity exists, the service returns either 204 No Content or 404 Not Found.

Example 66: collection with an entity reference for each Order related to the Product with `ID=0`

```
GET http://host/service/Products(0)/Orders/$ref
```

11.2.9 Resolving an Entity-Id

To resolve an entity-id, e.g. obtained in an entity reference, into a representation of the identified entity, the client issues a `GET` request to the `$entity` resource located at the URL `$entity` relative to the service root. The entity-id MUST be specified using the system query option `$id`.

Example 67: return the entity representation for a given entity-id

```
GET http://host/service/$entity?$id=http://host/service/Products(0)
```

A type segment following the `$entity` resource casts the resource to the specified type. If the identified entity is not of the specified type, or a type derived from the specified type, the service returns 404 Not Found.

After applying a type-cast segment to cast to a specific type, the system query options `$select` and `$expand` can be specified in `GET` requests to the `$entity` resource.

Example 68: return the entity representation for a given entity-id and specify properties to return

```
GET http://host/service/$entity/Model.Customer
      ?$id=http://host/service/Customers('ALFKI')
      &$select=CompanyName,ContactName
      &$expand=Orders
```

11.2.10 Requesting the Number of Items in a Collection

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

On success, the response body MUST contain the exact count of items matching the request after applying any `$filter` or `$search` system query options, formatted as a simple primitive integer value with media type `text/plain`. Clients SHOULD NOT combine the system query options `$top`, `$skip`, `$orderby`, `$expand`, and `$format` with the path suffix `/$count`. The result of such a request is undefined.

Example 69: return the number of products in the Products entity set

```
GET http://host/service/Products/$count
```

With 4.01 services the `/$count` segment MAY be used in combination with the `/$filter` path segment to count the items in the filtered collection.

Example 70: return the number of products whose `Price` *is less than $10.00*

```
GET http://host/service/Products/$filter(@foo)/$count?@foo=Price lt 10.00
```

For backwards compatibility, the `/$count` suffix MAY be used in combination with the `$filter` system query option.

Example 71: return the number of products whose `Price` is less than $10.00

```
GET http://host/service/Products/$count?$filter=Price lt 10.00
```

The `$filter` system query option MUST NOT be used in conjunction with a both a `/$count` path segment and a `/$filter` path segment.

The `/$count` suffix can also be used in path expressions within system query options, e.g. `$filter`.

Example 72: return all customers with more than five interests

```
GET http://host/service/Customers?$filter=Interests/$count gt 5
```

Example 73: return all categories with more than one product over $5.00

```
GET http://host/service/Categories?$filter=Products/$filter(Price gt 5.0)/$count gt 1
```

11.2.11 System Query Option `$format`

The `$format` system query option specifies the media type of the response.

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

The value of the `$format` system query option is a valid internet media type, optionally including parameters.

In addition, format-specific abbreviations may be used, e.g. `json` for `application/json`, see **[OData-JSON]**, but format parameters MUST NOT be appended to the format abbreviations.

Example 74: the request

```
GET http://host/service/Orders?$format=application/json;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 defined in **[OData-JSON]**.

Example 75: the request

```
GET http://host/service/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 defined in **[OData-JSON]**.

In metadata document requests, the values `application/xml` and `application/json`, along with their subtypes and parameterized variants, as well as the format-specific abbreviations `xml` and `json`, are reserved for this specification.

11.2.12 System Query Option `$schemaversion`

The `$schemaversion` system query option MAY be included in any request. For a metadata document request the value of the `$schemaversion` system query option addresses a specific schema version. For all other request types the value specifies the version of the schema against which the request is made. The syntax of the `$schemaversion` system query option is defined in **[OData-ABNF]**.

The value of the `$schemaversion` system query option MUST be a version of the schema as returned in the `Core.SchemaVersion` annotation, defined in **[OData-VocCore]**, of a previous request to the metadata document, or `*` in order to specify the current version of the metadata.

If specified, the service MUST process the request according to the specified version of the metadata.

Clients can retrieve the current version of the metadata by making a metadata document request with a `$schemaversion` system query option value of `*`, and SHOULD include the value from the returned `Core.SchemaVersion` annotation in the `$schemaversion` system query option of subsequent requests.

If the `$schemaversion` system query option is not specified in a request for the metadata document, the service MUST return a version of the metadata with no breaking changes over time, and the processing of all other requests that omit the `$schemaversion` system query option MUST be compatible with that "unversioned" schema. For more information on breaking changes, see Model Versioning.

If the `$schemaversion` system query option is specified on an individual request within a batch, then it specifies the version of the schema to apply to that individual request. Individual requests within a batch that don't include the `$schemaversion` system query option inherit the schema version of the overall batch request.

If the `$schemaversion` system query option is specified, but the version of the schema doesn't exist, the request is answered with a response code `404 Not Found`. The response body SHOULD provide additional information.

11.3 Requesting Changes

Services advertise their change-tracking capabilities by annotating entity sets with the `Capabilities.ChangeTracking` term defined in **[OData-VocCap]**.

Any `GET` request to retrieve one or more entities MAY allow change-tracking.

Clients request that the service track changes to a result by specifying the `track-changes` preference on a request. If supported for the request, the service includes a `Preference-Applied` header in the response containing the `track-changes` preference and includes a *delta link* in a result for a single entity, and on the last page of results for a collection of entities in place of the next link.

11.3.1 Delta Links

Delta links are opaque, service-generated links that the client uses 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 encodes the collection of entities for which changes are being tracked, along with a starting point from which to track changes. OData services may use the reserved system query option `$deltatoken` when building delta links. Its content is opaque, service-specific, and must only follow the rules for URL query parts.

If the defining query contains a `$schemaversion` system query option, the response MUST be represented according to that schema version.

If the defining query contains a `$filter` or `$search`, the response MUST include only changes to entities matching the specified criteria. Added entities MUST be returned for entities that were added or changed and now match the specified criteria, and deleted entities MUST be returned for entities that are changed to no longer match the criteria of `$filter` or `$search`.

The delta link MUST NOT encode any client top or skip value, and SHOULD NOT encode a request for an inline count.

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 links to expanded entities or nested collections representing current membership. If the defining query includes expanded references, then the delta link MUST return changes to the membership in the set of expanded references.

Navigation properties specified in the `$select` list of a defining query are not used to define the scope or contents of the items being tracked. Clients can specify `/$ref` in `$expand` in order to specify interest in the set of related entities without interest in changes to the content of those related entities.

If an expanded entity becomes orphaned because all paths to the entity as specified in the defining query have been broken (i.e. due to relationship changes and/or changes or deletions to parent entities) then the service MUST return the appropriate notifications for the client to determine that the entity has been orphaned (i.e. the changed relationships and removed parent entities). The client should not assume that it will receive additional notifications for such an orphaned entity.

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 only includes fields specified in the projection. Services MAY use the projection to limit the entities returned to those that have changed within the selected fields, but the client MUST be prepared to receive entities returned whether or not the field that changed was specified in the projection.

11.3.2 Using Delta Links

The client requests changes by invoking the `GET` method on the delta link. The client MUST NOT attempt to append system query options to the delta link. The `Accept` header MAY be used to specify the desired response format.

Clients SHOULD specify the same `Accept-Language` header when querying the delta link as was specified in the defining query. Services MAY return `406 Not Acceptable` if a different `Accept-Language` is specified. If a service does support an `Accept-Language` header it MAY return changes only visible in that language, or MAY include records that have changes not visible in the requested language.

The `/$count` segment can be appended to the path of a delta link in order to get just the number of changes available. The count includes all added, changed, or deleted entities, as well as added or deleted links.

The results of a request against the delta link may span one or more pages and MUST be ordered by the service across all pages in such a way as to guarantee consistency when applied in order to the response which contained the delta link.

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

In order to continue tracking changes beyond the current set, the client specifies `track-changes` on the initial request to the delta link but is not required to repeat it for subsequent pages. The new delta link appears at the end of the last page of changes in place of the next link and MUST return all changes subsequent to the last change of the previous delta link.

If no changes have occurred, the response is an empty collection that contains a delta link for subsequent changes if requested. This delta link MAY be identical to the delta link resulting in the empty collection of changes.

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

11.3.3 Delta Payloads

A delta payload represents changes to a known state. A delta payload includes added entities, changed entities, and deleted entities, as well as a representation of added and removed relationships.

Delta payloads can be requested from the service using a delta link or provided as updates to the service.

11.4 Data Modification

Updatable OData services support Create, Update, and Delete operations for some or all exposed entities. Additionally, Actions supported by a service can 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 `return` preference. A success response indicates that data have been modified, regardless of whether the requested content could be returned.

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

Each entity has its own ETag value that MUST change when structural properties or links from that entity have changed. In addition, modifying, adding, or deleting a contained entity MAY change the ETag of the parent entity.

Collections of entities (including collections of related entities) MAY have their own ETag value whose semantics is service-specific. It typically changes if entities are added to or removed from the collection, or if an entity in the collection is changed. The ETag of a collection of related entities reached via a navigation property MAY differ from the ETag of the entity containing the navigation property.

A Data Modification Request on an existing resource or an Action Request invoking an action bound to an existing resource MAY require optimistic concurrency control. Services SHOULD announce this via annotations with the terms `Core.OptimisticConcurrency` in **[OData-VocCore]** and `Capabilities.NavigationRestrictions` (nested property `OptimisticConcurrencyControl`) in **[OData-VocCap]**.

If optimistic concurrency control is required for a resource, the service MUST include an `ETag` header in a response to a `GET` request to the resource, and MAY include the ETag in a format-specific manner in responses containing that resource.

The presence of an `ETag` header in a response does not imply in itself that the resource requires optimistic concurrency control; the ETag may just be used for caching and/or conditional `GET` requests.

If an ETag value is specified in an `If-Match` or `If-None-Match` header of a Data Modification Request or Action Request, the operation MUST only be invoked if the `If-Match` or `If-None-Match` condition is satisfied.

If the client does not specify an `If-Match` request header in a Data Modification Request or Action Request on a resource that requires optimistic concurrency control, the service responds with a `428 Precondition Required` and MUST ensure that no observable change occurs as a result of the request. Clients can attempt to disable optimistic concurrency control by specifying `If-Match` with a value of `*`. Services MAY reject such requests.

For requests including an `OData-Version` header value of `4.01`, any ETag values specified in the request body of an update request MUST be `*` or match the current value for the record being updated.

11.4.1.2 Handling of DateTimeOffset Values

Services SHOULD preserve the offset of `Edm.DateTimeOffset` values, if possible. However, where the underlying storage does not support offset services may be forced to normalize the value to some common time zone (for example UTC) in which case the result would be returned with that time zone offset. If the service normalizes values, it MUST fail evaluation of the query functions `year`, `month`, `day`, `hour`, and `time` for literal values that are not stated in the time zone of the normalized values.

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 Integrity Constraints

Services may impose cross-entity integrity constraints. Certain referential constraints, such as requiring an entity to be created with related entities can be satisfied through creating or linking related entities when creating the entity. Other constraints might require multiple changes to be processed in an all-or-nothing fashion.

11.4.1.5 Returning Results from Data Modification Requests

Clients can request whether created or modified resources are returned from create, update, and upsert operations using the `return` preference header. In the absence of such a header, services SHOULD return the created or modified content unless the resource is a stream property value.

When returning content other than for an update to a media entity stream, services MUST return the same content as a subsequent request to retrieve the same resource. For updating media entity streams, the content of a non-empty response body MUST be the updated media entity.

Requests that return a single instance of a structured type or a collection of structured type instances MAY specify the system query options `$expand` and `$select`.

Requests that return a collection MAY specify the system query option `$filter`.

If one or more of these query options are present, this implies a `return=representation` preference if no `return` preference is specified.

If one or more of these query options are present with a `return=minimal` preference, the service SHOULD NOT return a representation and MUST include a `Preference-Applied` header if it does not return a representation.

If one or more of these query options are present and the service returns a representation, then the service MUST apply the specified query options. If it cannot apply the specified query options appropriately, it MUST NOT fail the request solely due to the presence of these query options and instead MUST return `204 No Content`.

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 representation of an entity of the declared target entity type, or one of its derived types.

The entity representation MAY include references to existing entities as well as content for new related entities, but MUST NOT contain content for existing related entities. The result of the operation is the entity with relationships to all included references to existing entities as well as all related entities created inline. If the key properties for an entity include key properties of a directly related entity, those related entities MUST be included either as references to existing entities or as content for new related entities.

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 new entity is automatically linked to the entity containing the navigation link.

If the target URL terminates in a type cast segment, then the segment MUST specify the type of, or a type derived from, the type of the collection, and the entity MUST be created as that specified type.

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.

Properties computed by the service (annotated with the term `Core.Computed`, see **[OData-VocCore]**) and properties that are tied to properties of the principal entity by a referential constraint, can be omitted and MUST be ignored if included in the request.

Properties with a defined default value, nullable properties, and collection-valued properties omitted from the request are set to the default value, null, or an empty collection, respectively.

Services MAY add dynamic properties to the created entity as long as their names do not conflict with the names of declared properties and client-specified dynamic properties.

Upon successful creation of the entity, the service MUST respond with either `201 Created` and a representation of the created entity, or `204 No Content` if the request included a `return=minimal` preference and did not include the system query options `$select` and `$expand`, or if a representation of the created entity could not be constructed. In either case, if the service is able to construct the edit URL or read URL of the created entity, the response MUST contain that URL in a `Location` header.

11.4.2.1 Link to Related Entities When Creating an Entity

To create a new entity with links to existing entities in a single request, the client includes references to the related entities in the request body.

The representation for referencing related entities is format-specific.

Example 76: using the JSON format, 4.0 clients can create a new manager entity with links to an existing manager (of managers) and to two existing employees by applying the `odata.bind` *annotation to the* `Manager` *and* `DirectReports` *navigation properties*

```
{
  "@type": "#Northwind.Manager",
  "ID": 1,
  "FirstName": "Pat",
  "LastName": "Griswold",
  "Manager@odata.bind": "http://host/service/Employees(0)",
  "DirectReports@odata.bind": [
    "http://host/service/Employees(5)",
    "http://host/service/Employees(6)"
  ]
}
```

Example 77: using the JSON format, 4.01 clients can create a new manager entity with links to an existing manager (of managers) and to two existing employees by including the entity-ids within the `Manager` *and* `DirectReports` *navigation properties*

```
{
  "@type": "#Northwind.Manager",
  "ID": 1,
  "FirstName": "Pat",
  "LastName": "Griswold",
  "Manager": { "@id": "Employees(0)" },
  "DirectReports": [
    {"@id": "Employees(5)"},
    {"@id": "Employees(6)"}
  ]
}
```

Upon successful completion of the operation, the service creates the requested entity and relates it to the requested existing entities.

If the target URL for the collection the entity is created in and binding information provided in the `POST` body contradicts the implicit binding information provided by the request URL, the request MUST fail, and the service responds with `400 Bad Request`.

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 request to create an entity that includes related entities, represented using the appropriate inline representation, is referred to as a “deep insert”.

Media entities MUST contain the format-specific representation of their media stream as a virtual property `$value` when nested within a deep insert.

Each included related entity is processed observing the rules for creating an entity as if it was posted against the original target URL extended with the navigation path to this related entity.

On success, the service MUST create all entities and relate them. If the service responds with `201 Created`, the response MUST be expanded to at least the level that was present in the deep-insert request.

Clients MAY associate an id with individual nested entities in the request by applying the `Core.ContentID` term using the namespace or alias defined for the **[OData-VocCore]** vocabulary in the service’s `$metadata` document. Services that respond with `201 Created` SHOULD annotate the entities in the response using the same `Core.ContentID` value as specified in the request. Services SHOULD advertise support for deep inserts, including support for returning the `Core.ContentID`, through the `Capabilities.DeepInsertSupport` term, defined in **[OData-VocCap]**; services that advertise support through `Capabilities.DeepInsertSupport` MUST return the `Core.ContentID` for the inserted or updated entities.

The `continue-on-error` preference is not supported for deep insert operations.

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

11.4.3 Update an Entity

To update an individual entity, the client makes a `PATCH` or `PUT` request to a URL that identifies the entity. Services MAY restrict updates only to requests addressing the edit URL of the entity.

The body of the request MUST be a valid representation of the declared target entity type, or one of its derived types.

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]**, is 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 corresponding to updatable properties MUST replace the value of the corresponding property in the entity or complex type. Complex properties are updated by applying `PATCH` semantics recursively, see also section 11.4.9.3. Omitted properties of the containing entity or complex property, including dynamic properties, MUST NOT be directly altered unless as a side effect of changes resulting from the provided properties.

If the type of the entity in a `PATCH` request differs from the type of the entity being updated (i.e., a different derived type of the declared target type), then properties shared through inheritance, as well as dynamic properties, are retained (unless overwritten by new values in the payload). Other properties of the original type are discarded.

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 values of structural properties with those specified in the request body. Omitted non-key, updatable structural properties not defined as dependent properties within a referential constraint MUST be set to their default values. Omitting a non-nullable property with no service-generated or default value from a `PUT` request results in a `400 Bad Request` error. Omitted dynamic properties MUST be removed, set to `null`, or set to a service-generated value.

For requests with an `OData-Version` header with a value of `4.01` or greater, the media stream of a media entity can be updated by specifying the format-specific representation of the media stream as a virtual property `$value`.

Updating a dependent property that is tied to a key property of the principal entity through a referential constraint updates the relationship to point to the entity with the specified key value. If there is no such entity, the update fails.

Updating a principal property that is tied to a dependent entity through a referential constraint on the dependent entity updates the dependent property.

Key and other properties marked as read-only in metadata (including computed properties), as well as dependent properties that are not tied to key properties of the principal entity, can be omitted from the request. If the request contains a value for one of these properties, the service MUST ignore that value when applying the update. Services MUST return an error if an insert or update contains a new value for a property marked as updatable that cannot currently be changed by the user (i.e., given the state of the object or permissions of the user). The service MAY return success in this case if the specified value matches the value of the property. Clients SHOULD use `PATCH` and specify only those properties intended to be changed.

The entity-id cannot be changed when updating an entity. However, format-specific rules might in some cases require providing the entity-id in the payload when requesting the update.

For requests with an `OData-Version` header with a value of `4.01` or greater, if the entity representation in the request body includes an ETag value, the update MUST NOT be performed and SHOULD return `412 Precondition Failed` if the supplied ETag value is not `*` and does not match the current ETag value for the entity. ETag values in request bodies MUST be ignored for requests containing an OData-Version header with a value of `4.0`.

If an update specifies both a binding to a single-valued navigation property and a dependent property that is tied to a key property of the principal entity according to the same navigation property, then the dependent property is ignored, and the relationship is updated according to the value specified in the binding.

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 MUST fail if it is unable to persist all updatable property values specified in the request.

Upon successful completion of the update, the service responds with either `200 OK` and a representation of the updated entity, or `204 No Content`. The client may request that the response SHOULD include a body by specifying a `return=representation` preference, or by specifying the system query options `$select` or `$expand`. If the service uses ETags for optimistic concurrency control, the entities in the response MUST include ETags. If a representation of the updated entity could not be constructed, the service MAY ignore the system query options and respond with `204 No Content`.

11.4.3.1 Update Related Entities When Updating an Entity

Update requests with an OData-Version header with a value of `4.0` MUST NOT contain related entities as inline content. Such requests MAY contain binding information for navigation properties. For single-valued navigation properties this replaces the relationship. For collection-valued navigation properties this adds to the relationship.

Payloads with an `OData-Version` header with a value of `4.01` or greater MAY include nested entities and entity references that specify the full set of to be related entities, or a nested delta payload representing the related entities that have been added, removed, or changed. Such a request is referred to as a “deep update”. If the nested collection is represented identical to an expanded navigation property, then the set of nested entities and entity references specified in a successful update request represents the full set of entities to be related according to that relationship and MUST NOT include added links, deleted links, or deleted entities.

Example 78: using the JSON format, a 4.01 `PATCH` *request can update a manager entity. Following the update, the manager has three direct reports; two existing employees and one new employee named* `Suzanne Brown`*. The* `LastName` *of employee 6 is updated to* `Smith`*.*

```
{
  "@type": "#Northwind.Manager",
  "FirstName": "Patricia",
  "DirectReports": [
    {
      "@id": "Employees(5)"
    },
    {
      "@id": "Employees(6)",
      "LastName": "Smith"
    },
    {
      "FirstName": "Suzanne",
      "LastName": "Brown"
    }
  ]
}
```

If the nested collection is represented as a delta annotation on the navigation property, then the collection contains members to be added or changed and MAY include deleted entities for entities that are no longer part of the collection, using the delta payload format. If the deleted entity specifies a `reason` as `deleted`, then the entity is both removed from the collection and deleted, otherwise it is removed from the collection and only deleted if the relationship is contained. Non-key properties of the deleted entity are ignored. Nested collections MUST NOT contain added or deleted links. If the request contains nested delta collections, then the `PATCH` verb must be specified.

If a nested entity has the same id or key fields as an existing entity, the existing entity is updated according to the semantics of the `PUT` or `PATCH` request. Nested entities that have no id or key fields, or for which the id or key fields do not match existing entities, are treated as inserts. If the nested collection does not represent a containment relationship and has no navigation property binding, then such entities MUST include a context URL specifying the entity set in which the new entity is to be created. If any nested entities contain both id and key fields, they MUST identify the same entity, or the request is invalid.

Example 79: using the JSON format, a 4.01 `PATCH` *request can specify a nested delta representation to:*

- *delete employee 3 and remove link to it*
- *remove the link to employee 4 and do not delete it*
- *add a link to employee 5*
- *change the last name of employee 6 and link to it if necessary*
- *add a new employee named “Suzanne Brown” and link to it*

```
{
  "@type": "#Northwind.Manager",
  "FirstName": "Patricia",
  "DirectReports@delta": [
    {
      "@removed": {
        "reason": "deleted"
      },
      "@id": "Employees(3)"
    },
    {
      "@removed": {
        "reason": "changed"
      },
      "@id": "Employees(4)"
    },
    {
      "@id": "Employees(5)"
    },
```

```
    {
      "@id": "Employees(6)",
      "LastName": "Smith"
    },
    {
      "FirstName": "Suzanne",
      "LastName": "Brown"
    }
  ]
}
```

Clients MAY associate an id with individual nested entities in the request by using the `Core.ContentID` term defined in **[OData-VocCore]**. Services that respond with `200 OK` SHOULD annotate the entities in the response using the same `Core.ContentID` value as specified in the request. Services SHOULD advertise support for deep updates, including support for returning the `Core.ContentID`, through the `Capabilities.DeepUpdateSupport` term, defined in **[OData-VocCap]**.

The `continue-on-error` preference is not supported for deep update operations.

On failure, the service MUST NOT apply any of the changes specified in the request.

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 yet exist. In this case the service MUST handle the request as a create entity request or fail the request altogether.

Upserts to single-valued navigation properties are possible for

- containment navigation properties,
- non-containment navigation properties with a navigation property binding, or
- payloads including a context URL specifying the entity set or contained collection of entities in which the new entity is to be created.

Upserts are not supported against entities whose keys' values are generated by the service. Services MUST fail an update request to a URL that would identify such an entity and the entity does not yet exist.

Similarly, services MUST fail an update request to the URL of a media entity that does not yet exist. However, a `PUT` request to the *media edit URL* of a media entity does have Upsert semantics, in that the media entity is created with the specified media stream if it does not already exist, otherwise the media stream of the existing media entity is updated.

Singleton entities can be upserted if they are nullable. Services supporting this SHOULD advertise it by annotating the singleton with the term `Capabilities.UpdateRestrictions` (nested property `Upsertable` with value `true`) defined in **[OData-VocCap]**.

Key and other non-updatable properties, as well as dependent properties that are not tied to key properties of the principal entity, MUST be ignored by the service in processing the Upsert request.

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 individual entity, the client makes a `DELETE` request to a URL that identifies the entity. Services MAY restrict deletes only to requests addressing the edit URL of the entity.

The request body SHOULD be empty. Top-level singleton entities can be deleted if they are nullable. Services supporting this MAY advertise it by annotating the singleton with the term `Capabilities.DeleteRestrictions` (nested property `Deletable` with value `true`) defined in **[OData-VocCap]**.

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.

One such integrity constraint results from using a navigation property in a key definition of an entity type. If the related “key” entity is deleted, the dependent entity is also deleted.

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 Add a Reference to a Collection-Valued Navigation Property

A successful `POST` request to a navigation property’s references collection adds a relationship to an existing entity. The request body MUST contain a single entity reference that identifies the entity to be added. See the appropriate format document for details.

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

Note that if the two entities are already related prior to the request, the request is completed successfully.

11.4.6.2 Remove a Reference to an Entity

A successful `DELETE` request to the URL that represents a reference to a related entity removes the relationship to that entity.

In OData 4.0, the entity reference to be removed within a collection-valued navigation property is the URL that represents the collection of related references, with the reference to be removed identified by the `$id` query option. OData 4.01 services additionally support using the URL that represents the reference of the collection member to be removed, identified by key, as described in **[OData-URL]**.

For single-valued navigation properties, the `$id` query option MUST NOT be specified.

The `DELETE` request MUST 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

A successful `PUT` request to a single-valued navigation property’s reference resource changes the related entity. The request body MUST contain a single entity reference that identifies the existing entity to be related. See the appropriate format document for details.

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

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 is format-specific, see **[OData-JSON]** for details.

If the single-valued navigation property is used in the key definition of an entity type, it cannot be changed and the request MUST fail with `405 Method Not Allowed` or an other appropriate error.

11.4.6.4 Replace all References in a Collection-Valued Navigation Property

A successful `PUT` request to a collection-valued navigation property's reference resource replaces the set of related entities. The request body MUST contain a collection of entity references in the same format as returned by a `GET` request to the navigation property's reference resource.

A successful `DELETE` request to a collection-valued navigation property's reference resource removes all related references from the collection.

11.4.7 Managing Media Entities

A media entity MUST have a source URL that can be used to read the media stream, and MAY have a media edit 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 Create a Media Entity

A `PUT` request to the media edit URL of a null-valued singleton media entity (by convention, the resource path of the media entity URL appended with `/$value`), or a `POST` request to a media entity's entity set, create a new media entity. The request body MUST contain the media value (for example, the photograph) whose media type MUST be specified in a `Content-Type` header. The request body is always interpreted as the media value, even if it has the media type of an OData format supported by the service. The service may set other structural properties of the media entity upon creation, but it is not possible for clients to specify structural properties when creating the media entity.

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

Upon successful completion the service responds with either `201 Created`, or `204 No Content` if the request included a `return=minimal` preference.

11.4.7.2 Update a Media Entity Stream

A successful `PUT` request to the media edit URL of an existing media entity changes the media stream of the entity.

If the media entity did not previously exist, then the request is interpreted as a creation request according to Create a 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.

The service may set other structural properties of the media entity when updating the media entity stream, but it is not possible for clients to specify structural properties when updating the media entity stream.

On success, the service MUST respond with either `204 No Content` and an empty body, or `200 OK` if the client specified the preference `return=representation`, in which case the response body MUST contain the updated media entity.

11.4.7.3 Delete a Media Entity

A successful `DELETE` request to the entity's edit URL or to the edit URL of its media stream deletes the media entity as described in Delete an Entity.

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

11.4.8 Managing Stream Properties

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

The values for stream properties do not usually appear in the entity payload unless explicitly requested with `$expand`. Instead, the values are generally read or written through URLs.

Example 80: read an entity and select a stream property

```
GET http://host/service/Products(1)?$select=Thumbnail
```

would only include control information for the stream property, not the stream data itself

```
{
  "@context": "http://host/service/$metadata#Products/$entity",
  …
  "Thumbnail@mediaReadLink": "http://server/Thumbnail546.jpg",
  "Thumbnail@mediaEditLink": "http://server/uploads/Thumbnail546.jpg",
  …
}
```

The stream data can then be requested using the media read link:

```
GET http://server/Thumbnail546.jpg
```

Services SHOULD support direct property access to a stream property's canonical URL. The response MAY be a redirect to the media read link of the stream property if the media read link is different from the canonical URL.

Example 81: directly read a stream property of an entity

```
GET http://host/service/Products(1)/Thumbnail
```

can return `200 OK` and the stream data (see section 11.2.4.1), or a `3xx Redirect` to the media read link of the stream property.

Note: for scenarios in which the media value can only be inlined, the property should instead be modeled with type `Edm.Binary`.

11.4.8.1 Update Stream Values

A successful `PUT` request to the edit URL of a stream property changes the media stream associated with that property.

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. It may have a `Content-Length` of zero to set the stream data to empty.

Stream properties MAY specify a list of acceptable media types using an annotation with term `Core.AcceptableMediaTypes`, see **[OData-VocCore]**.

On success, the service MUST respond with either `204 No Content` and an empty body, or `200 OK` if the client specified the preference `return=representation`, in which case the response body MUST contain the updated media value for the stream.

Clients MAY change the association between a stream property and a media stream by modifying the edit URL or read URL of the stream property. Services supporting this SHOULD advertise it by annotating the stream property with the term `Capabilities.MediaLocationUpdateSupported` defined in **[OData-VocCap]**.

11.4.8.2 Delete Stream Values

A successful `DELETE` request to the edit URL of a stream property attempts to set the property to null and results in an error if the property is non-nullable.

Example 82: delete the stream value using the media edit link retrieved in example 80

```
DELETE http://server/uploads/Thumbnail546.jpg
```

Attempting to request a stream property whose value is null results in `204 No Content`.

11.4.9 Managing Values and Properties Directly

Values and properties can be explicitly addressed with URLs. The edit URL of a property is the edit URL of the entity appended with the path segment(s) specifying the individual property. The edit URL allows properties to be individually modified. See **[OData-URL]** for details on addressing individual properties.

11.4.9.1 Update a Primitive Property

A successful `PUT` request to the edit URL for a primitive property updates the value of the property. The message body MUST contain the new value, formatted as a single property according to the specified format.

A successful `PUT` request to the edit URL for the raw value of a primitive property updates the property with the raw value specified in the payload. The payload MUST be formatted as an appropriate content type for the raw value of the property.

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

Upon successful completion the service responds with either `200 OK` or `204 No Content`. The client may request that the response SHOULD include a body by specifying a `return=representation` preference.

Services MUST return an error if the property is not updatable.

11.4.9.2 Set a Value to Null

A successful `DELETE` request to the edit URL for a structural property, or to the edit URL of the raw value of a primitive property, sets the property to null. The request body is ignored and should be empty.

A `DELETE` request to a non-nullable value MUST fail and the service respond with `400 Bad Request` or other appropriate error.

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.

Services MUST return an error if the property is not updatable.

Updating a primitive property or a complex property with a null value also sets the property to null.

11.4.9.3 Update a Complex Property

A successful `PATCH` request to the edit URL for a complex typed property updates that property. The request body MUST contain a single valid representation for the declared type of the complex property or one of its derived types.

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

If a complex-typed property is set to a different type in a `PATCH` request, properties shared through inheritance, as well as dynamic properties, are retained (unless overwritten by new values in the payload). Other properties of the original type are discarded.

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.

Upon successful completion the service responds with either `200 OK` or `204 No Content`. The client may request that the response SHOULD include a body by specifying a `return=representation` preference.

Services MUST return an error if the property is not updatable.

11.4.9.4 Update a Collection Property

A successful `PUT` request to the edit URL of a collection property updates that collection. 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.

A successful `POST` request to the edit URL of a collection property adds an item to the collection. The body of the request MUST be a single item to be added to the collection. If the collection is ordered, the item is added to the end of the collection, and if the collection supports positional insert `$index` MAY be used to specify the insert position.

A successful `DELETE` request to the edit URL of a collection property deletes all items in that collection.

Since collection members have no individual identity, `PATCH` is not supported for collection properties.

Upon successful completion the service responds with either `200 OK` and a representation of the updated collection, or `204 No Content`. The client may request that the response SHOULD include a body by specifying a `return=representation` preference.

Services MUST return an error if the property is not updatable.

11.4.10 Managing Members of an Ordered Collection

Collections annotated with the `Core.Ordered` term (see **[OData-VocCore]**) have a stable order. Members of an ordered collection of primitive and complex types can be individually updated or deleted by invoking an update operation against the URL of the collection appended by a segment containing the zero-based ordinal of the item within the collection. A negative ordinal number indexes from the end of the collection, with -1 representing the last item in the collection.

Entities can be updated using their edit URL and SHOULD NOT be addressed using an index.

11.4.11 Positional Inserts

Collections of entity, complex, or primitive types annotated with the `Core.PositionalInsert` term (see **[OData-VocCore]**) support inserting items at a specific location via `POST` requests to the collection URL using the `$index` system query option. The value of the `$index` system query option is the zero-based ordinal position where the item is to be inserted. The ordinal positions of items within the collection greater than or equal to the inserted position are increased by one. A negative ordinal number indexes from the end of the collection, with -1 representing an insert as the last item in the collection.

Example 83: Insert a new email address at the second position

```
POST /service/Customers('ALFKI')/EmailAddresses?$index=1
Content-Type: application/json

{
```

```
  "value": "alfred@futterkiste.de"
}
```

11.4.12 Update a Collection of Entities

Collections of entities can be updated by submitting a `PATCH` request to the resource path of the collection. The body of the request MUST be a delta payload, and the resource path of the collection MUST NOT contain type cast or filter segments, and MUST NOT contain any system query options that affect the shape of the result.

Added/changed entities are applied as upserts, and deleted entities as deletions. Non-key properties of deleted entities are ignored. The top-level collection may include added and deleted links, and related entities represented inline are updated according to the rules for treating related entities when updating an entity.

Clients MAY associate an id with individual nested entities in the request by using the `Core.ContentID` term defined in **[OData-VocCore]**. Services that respond with `200 OK` SHOULD annotate the entities in the response using the same `Core.ContentID` value as specified in the request.

Services SHOULD advertise support for updating a collection using a delta payload through the `DeltaUpdateSupported` property of the `Capabilities.UpdateRestrictions` term, and SHOULD advertise support for returning the `Core.ContentID` through the `ContentIDSupported` property of the `Capabilities.DeepUpdateSupport` term, both defined in **[OData-VocCap]**.

The response, if requested, is a delta payload, in the same structure and order as the request payload, representing the applied changes.

If the `continue-on-error` preference has been specified and any errors occur in processing the changes, then a delta response MUST be returned regardless of the `return` preference and MUST contain at least the failed changes. The service represents failed changes in the delta response as follows:

- Failed deletes in the request MUST be represented in the response as either entities or entity references, annotated with the term `Core.DataModificationException`, see **[OData-VocCore]**. If the deleted entity specified a reason of `deleted`, the value of `failedOperation` MUST be `delete`, otherwise `unlink`.
- Failed inserts within the request MUST be represented in the response as deleted entities annotated with the term `Core.DataModificationException` with a `failedOperation` value of `insert`.
- Failed updates within the request SHOULD be annotated in the response with the term `Core.DataModificationException` with a `failedOperation` value of `update`.
- Failed added links within the request MUST represented in the response as deleted links annotated with the term `Core.DataModificationException` with a `failedOperation` value of `link`.
- Failed deleted links within the request MUST represented in the response as added links annotated with the term `Core.DataModificationException` with a `failedOperation` value of `unlink`.
- Collections within the request MUST be represented in the response as a collection with the current values and membership of the collection as it exists in the service after processing the request.

If an individual change fails due to a failed dependency, it MUST be annotated with the term `Core.DataModificationException` and SHOULD specify a `responseCode` of `424` (Failed Dependency).

Alternatively, the verb `PUT` can be used, in which case the request body MUST be the representation of a collection of entities. In this case all entities provided in the request are applied as upserts, and any entities not provided in the request are deleted. In this case, if the `continue-on-error` preference has been specified, and the request returns a success response code, then a response MUST be returned regardless of the `return` preference, and MUST contain the full membership and values of the collection as it exists in the service.

If the `continue-on-error` preference has not been specified, and the service is unable to apply all of the changes in the request, then it MUST return an error response and MUST NOT apply any of the changes specified in the request payload.

11.4.13 Update Members of a Collection

Members of a collection can be updated by submitting a `PATCH` request to the URL constructed by appending `/$each` to the resource path of the collection. The additional path segment expresses that the request body describes an update to each member of the collection, not an update to the collection itself.

The resource path of the collection MAY contain type-cast or filter segments to subset the collection, see **[OData-URL]**.

For primitive-typed collections the body of the request MUST be a primitive value. Each member of the potentially filtered collection is updated to the specified primitive value.

For collections of structured type, the body of the request MUST be a full or partial representation of an instance of the collection's structured type. Each member of the potentially filtered collection is updated using `PATCH` semantics. Structured types MAY include nested collections or delta collections, in which case the semantics described in Update a Collection of Entities applies.

Example 84: change the color of all beige-brown products

```
PATCH /service/Products/$filter(@bar)/$each?@bar=Color eq 'beige-brown'
Content-Type: application/json

{
  "Color": "taupe"
}
```

The response, if requested, is a collection payload containing the updated representation of each member identified by the request. If the update payload includes nested collections or nested delta collections, then they MUST be included in the response, as described in Update a Collection of Entities.

Clients should note that requesting a response may be expensive for services that could otherwise efficiently apply updates to a (possibly filtered) collection.

If the `continue-on-error` preference has been specified, the service MAY continue processing updates after a failure. In this case, the service MUST return a response containing at least the members of the collection that failed to update, which MUST be annotated with the term `Core.DataModificationException` with a `failedOperation` value of `update`.

If the `continue-on-error` preference has not been specified, and the service is unable to update all of the members identified by the request, then it MUST return an error response and MUST NOT apply any updates.

11.4.14 Delete Members of a Collection

Members of a collection can be deleted by submitting a `DELETE` request to the URL constructed by appending `/$each` to the resource path of the collection. The additional path segment expresses that the collection itself is not deleted.

The request resource path of the collection MAY contain type-cast or filter segments to subset the collection.

Example 85: delete all products older than 3

```
DELETE /service/Products/$filter(Age gt 3)/$each
```

If the path identifies a collection of entities and if the service returns a representation, then the response is a delta response containing a representation of a deleted entity for each deleted member.

If the collection is a collection of entities, then the client MAY specify the `continue-on-error` preference, in which case the service MAY continue processing deletes after a failure. In this case, the service MUST return a response containing at least an entity or entity reference for each entity identified by the request that failed to delete, which MUST be annotated with the term `Core.DataModificationException` with a `failedOperation` value of `delete`.

Clients should note that requesting a response may be expensive for services that could otherwise efficiently apply deletes to a (possibly filtered) collection.

If the `continue-on-error` preference has not been specified, and the service is unable to delete all of the entities identified by the request, then it MUST return an error response and MUST NOT apply any changes.

11.5 Operations

Custom operations (Actions and Functions) allow encapsulating logic for modifying or requesting data that goes beyond simple CRUD described in the preceding sections of this chapter. See `Action`, `ActionImport`, `Function`, and `FunctionImport` in **[OData-CSDLJSON]** or **[OData-CSDLXML]**.

11.5.1 Binding an Operation to a Resource

Actions and Functions MAY be bound to any type or collection, similar to defining a method in a class in object-oriented programming. The first parameter of a bound operation is the *binding parameter*.

The namespace- or alias-qualified name of a bound 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*. Only aliases defined in the metadata document of the service can be used in URLs.

Example 86: the function `MostRecentOrder` *can be bound to any URL that identifies a* `SampleModel.Customer`

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

Example 87: invoke the `MostRecentOrder` *function with the value of the binding parameter* `customer` *being the entity identified by* `http://host/service/Customers(6)`

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

Example 88: the function `Comparison` *can be bound to any URL that identifies a collection of entities*

```
<Function Name="Comparison" IsBound="true">
  <Parameter Name="in" Type="Collection(Edm.EntityType)" />
  <ReturnType Type="Diff.Overview" />
</Function>
```

Example 89: invoke the `Comparison` *function on the set of red products*

```
GET http://host/service/Products/$filter(Color eq 'Red')/Diff.Comparison()
```

11.5.2 Applying an Operation to Members of a Collection

A bound operation with a single-valued binding parameter can be applied to each member of a collection by appending the path segment `/$each` to the resource path of the collection, followed by a forward slash and the namespace- or alias-qualified name of the bound operation. In this case the type of the collection members MUST match or be derived from the type of the binding parameter.

The resource path of the collection MAY contain type-cast or filter segments to subset the collection.

The response is a collection with members that are instances of the result type of the bound operation. If the bound operation returns a collection, the response is a collection of collections.

Example 90: invoke the `MostRecentOrder` *function on each entity in the entity set* `Customers`

```
GET http://host/service/Customers/$each/SampleModel.MostRecentOrder()
```

The client MAY specify the `continue-on-error` preference, in which case the service MAY continue processing actions after a failure. In this case, the service MUST, regardless of the `return` preference, return a response containing at least the members identified by the request for which the action failed. Such members MUST be annotated with the term `Core.DataModificationException` with a `failedOperation` value of `invoke`.

If the `continue-on-error` preference has not been specified, and the service is unable to invoke the action against all of the entities identified by the request, then it MUST return an error response and MUST NOT apply the action to any of the members of the collection.

11.5.3 Advertising Available Operations within a Payload

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

Example 91: given a `GET` *request to* `http://host/service/Customers('ALFKI')`*, the service might respond with a Customer that includes the* `SampleEntities.MostRecentOrder` *function bound to the entity*

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

An efficient format that assumes client knowledge of metadata may omit actions and functions from the payload whose target URL can be computed via metadata following standard conventions defined in **[OData-URL]**.

Services can advertise that a function or action is not available for a particular instance by setting its value to null.

Example 92: the `SampleEntities.MostRecentOrder` *function is not available for customer* `ALFKI`

```
{
  "@context": …,
  "CustomerID": "ALFKI",
  "CompanyName": "Alfreds Futterkiste",
  "#SampleEntities.MostRecentOrder": null,
  …
}
```

11.5.4 Functions

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

11.5.4.1 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 representation 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 derived 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 inline parameter syntax. In parameters containing complex instances or transient entities, properties with a defined default value,

nullable properties, and collection-valued properties that are omitted from the request are interpreted as the default value, null, or an empty collection, respectively.

If the function URL is obtained from a previously returned entity representation, parameter aliases that are identical to the parameter name preceded by an at (`@`) sign MUST be used. Clients MUST check if the obtained URL already contains a query part and appropriately precede the parameters either with an ampersand (`&`) or a question mark (`?`).

Services MAY additionally support invoking functions using the unqualified function name by defining one or more default namespaces through the `Core.DefaultNamespace` term defined in **[OData-VocCore]**.

To request processing of the function only if the binding parameter value, an entity or collection of entities, is unmodified, the client includes the `If-Match` header with the latest known ETag value for the entity or collection of entities. The ETag value for a collection as a whole is transported in the `ETag` header of a collection response.

Functions can be used within `$filter` or `$orderby` system query options. Such functions can 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` are specified according to the inline parameter syntax.

To invoke a function through a function import the client issues a `GET` request to a URL identifying the function import and passing parameter values using inline parameter syntax. The canonical URL for a function import is the service root, followed by the name of the function import. Services MAY support omitting the parentheses when invoking a function import with no parameters, but for maximum interoperability MUST also support invoking the function import with empty parentheses.

If the function is composable, additional path segments may be appended to the URL that identifies the composable function (or function import) as appropriate for the type returned by the function (or function import). The last path segment determines the system query options and HTTP verbs that can be used with this this URL, e.g. if the last path segment is a multi-valued navigation property, a `POST` request may be used to create a new entity in the identified collection.

Example 93: add a new item to the list of items of the shopping cart returned by the composable `MyShoppingCart` function import

```
POST http://host/service/MyShoppingCart()/Items

…
```

If the function returns a value of type `Edm.Stream` and no additional path segments follow the function invocation, the response to the `GET` request follows the rules for requesting stream properties.

Parameter values passed to functions MUST be specified either as a URL literal (for primitive values) or as a JSON formatted OData object (for complex values, or collections of primitive or complex values). Entity typed values are passed as JSON formatted entities that MAY include a subset of the properties, or just the entity reference, as appropriate to the function.

If a collection-valued function has no result for a given parameter value combination, the response is the format-specific representation of an empty collection. If a single-valued function with a nullable return-type has no result, the service returns `204 No Content`.

If a single-valued function with a non-nullable return type has no result, the service returns `4xx`. For functions that return a single entity `404 Not Found` is the appropriate response code.

For a composable function the processing is stopped when the function result requires a `4xx` response, and continues otherwise.

Function imports preceded by the `$root` literal MAY be used in the `$filter` or `$orderby` system query options, see **[OData-URL]**.

11.5.4.1.1 Inline Parameter Syntax

Parameter values are specified inline by appending a comma-separated list of parameter values, enclosed by parenthesis to the function name.

Each parameter value is 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.

Example 94: invoke a `Sales.EmployeesByManager` *function which takes a single* `ManagerID` *parameter via the function import* `EmployeesByManager`

```
GET http://host/service/EmployeesByManager(ManagerID=3)
```

Example 95: return all Customers whose `City` *property returns* `Western` *when passed to the* `Sales.SalesRegion` *function*

```
GET http://host/service/Customers?
      $filter=Sales.SalesRegion(City=$it/City) eq 'Western'
```

A parameter alias can be used in place of an inline parameter value. The value for the alias is specified as a separate query option using the name of the parameter alias.

Example 96: invoke a `Sales.EmployeesByManager` *function via the function import* `EmployeesByManager`*, passing 3 for the* `ManagerID` *parameter*

```
GET http://host/service/EmployeesByManager(ManagerID=@p1)?@p1=3
```

Services MAY in addition allow implicit parameter aliases for function imports and for functions that are the last path segment of the URL. An implicit parameter alias is the parameter name, optionally preceded by an at (`@`) sign. When using implicit parameter aliases, parentheses MUST NOT be appended to the function (import) name. The value for each parameter MUST be specified as a separate query option with the name of the parameter alias. If a parameter name is identical to a system query option name (without the optional `$` prefix), the parameter name MUST be prefixed with an at (`@`) sign.

Example 97: invoke a `Sales.EmployeesByManager` *function via the function import* `EmployeesByManager`*, passing 3 for the* `ManagerID` *parameter using the implicit parameter alias*

```
GET http://host/service/EmployeesByManager?ManagerID=3
```

Non-binding parameters annotated with the term `Core.OptionalParameter` defined in **[OData-VocCore]** MAY be omitted. If it is annotated and the annotation specifies a `DefaultValue`, the omitted parameter is interpreted as having that default value. If omitted and the annotation does not specify a default value, the service is free on how to interpret the omitted parameter.

11.5.4.2 Function overload resolution

The same function name may be used multiple times within a schema, each with a different set of parameters. For unbound overloads the combination of the function name and the unordered set of parameter names MUST identify a particular function overload. For bound overloads the combination of the function name, the binding parameter type, and the unordered set of names of the non-binding parameters MUST identify a particular function overload.

All unbound overloads MUST have the same return type. Also, all bound overloads with a given binding parameter type MUST have the same return type.

If the function is bound 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. A type-cast segment can be used to select a function defined on a particular type in the hierarchy, see **[OData-URL]**.

Non-binding parameters MAY be marked as optional by annotating them with the term `Core.OptionalParameter` defined in **[OData-VocCore]**. All parameters marked as optional MUST come after any parameters not marked as

optional.

A function overload is selected if

- The set of specified parameters exactly matches a function overload, or else
- The set of specified parameters matches a subset of parameters that includes all non-optional parameters of exactly one function overload.

Services SHOULD avoid ambiguity, i.e. the combination of the function name, the unordered set of *non-optional* non-binding parameter names, plus the binding parameter type for bound overloads SHOULD identify a particular function overload. If there is ambiguity, then services MAY return `400 Bad Request` with an error response body stating that the request was ambiguous.

11.5.5 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 with additional path segments.

11.5.5.1 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 representation or constructed by appending the namespace- or alias-qualified action 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 action. The value for the binding parameter is the resource identified by the URL preceding the action name, and only the non-binding parameter values are passed in the request body according to the particular format. In parameters containing complex instances or transient entities, properties with a defined default value, nullable properties, and collection-valued properties that are omitted from the request are interpreted as the default value, null, or an empty collection, respectively.

Services MAY additionally support invoking actions using the unqualified action name by defining one or more default namespaces through the `Core.DefaultNamespace` term defined in **[OData-VocCore]**.

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. When invoking an action through an action import all parameter values MUST be passed in the request body according to the particular format.

Non-binding parameters that are nullable or annotated with the term `Core.OptionalParameter` defined in **[OData-VocCore]** MAY be omitted from the request body. If an omitted parameter is not annotated (and thus nullable), it MUST be interpreted as having the `null` value. If it is annotated and the annotation specifies a `DefaultValue`, the omitted parameter is interpreted as having that default value. If omitted and the annotation does not specify a default value, the service is free on how to interpret the omitted parameter. Note: a nullable non-binding parameter is equivalent to being annotated as optional with a default value of `null`.

4.01 services MUST support invoking actions with no non-binding parameters and parameterless action imports both without a request body and with a request body representing no parameters, according to the particular format. Interoperable clients SHOULD always include a request body, even when invoking actions with no non-binding parameters and parameterless action imports.

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.

The client can request whether any results from the action be returned using the `return` preference.

Actions that create and return a single entity follow the rules for entity creation and return a `Location` header that contains the edit URL or read URL of the created entity. They MAY be annotated with the term `Core.Constructor` defined in **[OData-VocCore]**.

If the action returns a value of type `Edm.Stream`, the response to the `POST` request follows the rules for requesting stream properties.

Actions without a return type respond with `204 No Content` on success.

To request processing of the action only if the binding parameter value, an entity or collection of entities, is unmodified, the client includes the `If-Match` header with the latest known ETag value for the entity or collection of entities. The ETag value for a collection as a whole is transported in the `ETag` header of a collection response.

Example 98: invoke 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. Invoke the action only if the customer's ETag still matches.*

```
POST http://host/service/Customers('ALFKI')/SampleEntities.CreateOrder
If-Match: W/"MjAxOS0wMy0yMVQxMzowNVo="
Content-Type: application/json

{
  "items": [
    { "product": 4001, "quantity": 2 },
    { "product": 7062, "quantity": 1 },
  ],
  "discountCode": "BLACKFRIDAY"
}
```

11.5.5.2 Action Overload Resolution

The same action name may be used multiple times within a schema provided there is at most one unbound overload, and each bound overload specifies a different binding parameter type.

If the action is bound 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. A type-cast segment can be used to select an action defined on a particular type in the hierarchy, see **[OData-URL]**.

11.6 Asynchronous Requests

A `Prefer` header with a `respond-async` preference allows clients to request that the service process a Data Service 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. A service MUST NOT reply to a Data Service Request with `202 Accepted` if the request has not included the `respond-async` preference.

Responses that return `202 Accepted` MUST include a `Location` header pointing to a *status monitor resource* that represents the current state of the asynchronous processing in addition to an optional `Retry-After` header indicating the time, in seconds, the client should wait before querying the service for status. Services MAY include a response body, for example, to provide additional status information.

A `GET` request to the status monitor resource again returns `202 Accepted` response if the asynchronous processing has not finished. This response MUST again include a `Location` header and MAY include a `Retry-After` header to be used for a subsequent request. The `Location` header and optional `Retry-After` header may or may not contain the same values as returned by the previous request.

A `GET` request to the status monitor resource returns `200 OK` once the asynchronous processing has completed. For OData 4.01 or greater responses, or OData 4.0 requests that include an `Accept` header that does not specify `application/http`, the response MUST include the `AsyncResult` response header. Any other headers, along with the response body, represent the result of the completed asynchronous operation. If the `GET` request to the status monitor includes an `OData-MaxVersion` header with a value of `4.0` and no `Accept` header, or an `Accept` header that

includes `application/http`, then the body of the final `200 OK` response MUST be represented as an HTTP message, as described in **[RFC9110]**, which is the full HTTP response to the completed asynchronous operation.

A `DELETE` request sent to the status monitor resource requests that the asynchronous processing be canceled. A `200 OK` or a `204 No Content` response indicates that the asynchronous processing has been successfully canceled. A client can request that the `DELETE` should be executed asynchronously. A `202 Accepted` response indicates that the cancellation is being processed asynchronously; the client can use the returned `Location` header (which MUST be different from the status monitor resource of the initial request) to query for the status of the cancellation. If a delete request is not supported by the service, the service returns `405 Method Not Allowed`.

After a successful `DELETE` request against the status monitor resource, any subsequent `GET` requests for the same status monitor resource returns `404 Not Found`.

If an asynchronous request is cancelled for reasons other than the consumers issuing a `DELETE` request against the status monitor resource, a `GET` request to the status monitor resource returns `200 OK` with a response body containing a single HTTP response with a status code in the `5xx Server Error` range indicating that the operation was cancelled.

The service MUST ensure that no observable change has occurred as a result of a canceled request.

If the client waits too long to request the result of the asynchronous processing, the service responds with a `410 Gone` or `404 Not Found`.

The status monitor resource URL MUST differ from any other resource URL.

11.7 Batch Requests

Batch requests allow grouping multiple individual requests into a single HTTP request payload. An individual request in the context of a batch request is a Metadata Request, Data Request, Data Modification Request, Action invocation request, or Function invocation request.

Batch requests are submitted as a single HTTP `POST` request to the batch endpoint of a service, located at the URL `$batch` relative to the service root.

Individual requests within a batch request are evaluated according to the same semantics used when the request appears outside the context of a batch request.

A batch request is represented using either the multipart batch format defined in this document or the JSON batch format defined in **[OData-JSON]**.

If the set of request headers of a batch request are valid the service MUST return a `200 OK` HTTP response code to indicate that the batch request was accepted for processing, even if the processing is yet to be completed. The individual requests within the body of the batch request may be processed as soon as they are received, this enables clients to stream batch requests, and 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 batch request.

11.7.1 Batch Request Headers

A batch request using the multipart batch format MUST contain a `Content-Type` header specifying a content type of `multipart/mixed` and a `boundary` parameter as defined in **[RFC2046]**.

Example 99: multipart batch request

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

```
<Multipart Batch request body>
```

A batch request using the JSON batch format MUST contain a `Content-Type` header specifying a content type of `application/json`.

Example 100: JSON batch request

```
POST /service/$batch HTTP/1.1
Host: odata.org
OData-Version: 4.01
Content-Type: application/json

<JSON Batch request body>
```

Batch requests SHOULD contain the applicable `OData-Version` header.

Batch requests SHOULD contain an `Accept` header specifying the desired batch response format, either `multipart/mixed` or `application/json`. If no `Accept` header is provided, services SHOULD respond with the content type of the request.

11.7.2 Request Dependencies

Requests within a batch may have dependencies on other requests according to the particular batch format.

In the JSON format, requests may explicitly declare a dependency on other requests that must be successfully processed before the current request. In addition, requests may be specified as part of an *atomicity group* whose members MUST either all succeed, or all fail. If a request fails, then any dependent requests within the JSON format return `424 Failed Dependency`.

In the Multipart format, data modification requests or action invocation requests may be grouped as part of an atomic change set. Operations outside the change set are executed sequentially, while operations within the change set may be executed in any order.

11.7.3 Identifying Individual Requests

Each individual request within a batch request MAY have a request identifier assigned. The request identifier is case-sensitive, MUST be unique within the batch request, and MUST satisfy the rule `request-id` in **[OData-ABNF]**.

The representation of the request identifier is format-specific, as are the rules for which individual requests require an identifier.

11.7.4 Referencing Returned Entities

Entities created by an insert request or an action can be referenced in the request URL of subsequent requests by using the request identifier prefixed with a `$` character as the first segment of the request URL. Services MUST treat this segment like the URL in the `Location` header of the response to the request identified by the segment. If the `Location` header in the response to the subsequent request contains a relative URL, clients MUST be able to resolve it relative to the request's URL even if that contains such a reference. See example 105.

If the `$`-prefixed request identifier is identical to the name of a top-level system resource (`$batch`, `$crossjoin`, `$all`, `$entity`, `$root`, `$id`, `$metadata`, or other system resources defined according to the `OData-Version` of the protocol specified in the request), then the reference to the top-level system resource is used. This collision can be avoided by e.g. using only numeric request identifiers.

Services MAY also support referencing within request bodies, in which case they SHOULD advertise this support by specifying the `ReferencesInRequestBodiesSupported` property in the `Capabilities.BatchSupport` term applied to the entity container, see **[OData-VocCap]**.

11.7.5 Referencing the ETag of an Entity

Services MAY support the use of an ETag returned from a previous operation in an `If-Match` or `If-None-Match` header of a subsequent statement. Services SHOULD advertise this support by specifying the `EtagReferencesSupported` property in the `Capabilities.BatchSupport` annotation term applied to the entity container, see **[OData-VocCap]**.

The ETag for a previous operation can be referenced by using the request identifier prefixed with a `$` character as the unquoted value of the `If-Match` or `If-None-Match` header.

11.7.6 Referencing Values from Response Bodies

Services MAY support using values from a response body in the query part of the URL or in the request body of subsequent requests. Value references consist of a `$` character, followed by the identifier of the preceding request. They evaluate to the referenced value, that is the value represented by the response body of that preceding request. If that value is a collection, the value reference MAY be followed by a `collectionNavigationExpr`, as defined in **[OData-ABNF]**, that is evaluated relative to the referenced value. Otherwise the value reference MAY be followed by a forward slash and a `memberExpr` that is evaluated relative to the referenced value.

If the `$`-prefixed request identifier is identical to the name of a predefined literal for query expressions (`$it`, `$root`, or other literals defined according to the `OData-Version` of the protocol specified in the request), then the predefined literal is used. This collision can be avoided by e.g. using only numeric request identifiers.

11.7.7 Multipart Batch Format

The multipart batch format is represented as a Multipart Media Type message **[RFC2046]**, a standard format allowing the representation of multiple parts, each of which may have a different content type.

11.7.7.1 Multipart Batch Request Body

The body of a multipart batch request is made up of a series of individual requests and *change sets*, each represented as a distinct body part (i.e. preceded by a boundary delimiter line consisting of two dashes and the value of the `boundary` parameter specified in the `Content-Type` header, and the last body part followed by a closing boundary delimiter line consisting of two dashes, the boundary, and another two dashes).

A body part representing an individual request MUST include a `Content-Type` header with value `application/http`.

The contents of a body part representing a change set MUST itself be a multipart document (see **[RFC2046]**) with one body part for each operation in the change set. Each body part representing an operation in the change set MUST specify a `Content-ID` header with a request identifier that is unique within the batch request.

A `Content-Transfer-Encoding` header with value `binary` may be included for historic reasons although this header is not used by HTTP and only needed for transmission via E-Mail. Neither clients nor services should rely on this header being present.

Preambles and epilogues in the multipart batch request body, as defined in **[RFC2046]**, are valid but are assigned no meaning and thus MUST be ignored by processors of multipart batch requests.

The request URL of individual requests within a batch request or change set can use one of the following three formats:

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

Example 101:

```
GET https://host:1234/path/service/People(1) HTTP/1.1
```

- Absolute resource path and separate `Host` header

Example 102:

```
PATCH /path/service/People(1) HTTP/1.1
Host: myserver.mydomain.org:1234
Content-Type: application/json

{"Name": "Peter"}
```

- Resource path relative to the batch request URI.

Example 103:

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

Services MUST support all three formats for URLs of individual requests.

URLs must be correctly percent-encoded. For relative URLs this means that colons in the path part, especially within key values, MUST be percent-encoded to avoid confusion with the scheme separator. Colons within the query part, i.e. after the question mark character (`?`), need not be percent-encoded.

Each body part 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 body parts. For example, a processor may choose to disallow chunked encoding to be used by such HTTP requests.

Example 104: a batch request that contains the following individual requests in the order listed

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

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

*Note also that the two empty lines after the `Host` header of the `GET` request are necessary: the first is part of the `GET` request header; the second is the empty body of the `GET` request, followed by a `CRLF` according to **[RFC2046]**.*

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

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

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


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

--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
Content-Type: application/http
Content-ID: 1

POST /service/Customers HTTP/1.1
```

```
Host: host
Content-Type: application/json
Content-Length: ###

<JSON representation of a new Customer>
--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
Content-Type: application/http
Content-ID: 2

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

<JSON representation of changes to Customer ALFKI>
--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd--
--batch_36522ad7-fc75-4b56-8c71-56071383e77b
Content-Type: application/http

GET /service/Products HTTP/1.1
Host: host


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

11.7.7.2 Referencing New Entities

Entities created by an insert request or an action can be referenced in the request URL of subsequent requests within the same change set. Services MAY also support referencing across change sets, in which case they SHOULD advertise this support by specifying the `ReferencesAcrossChangeSetsSupported` property in the `Capabilities.BatchSupport` term applied to the entity container, see **[OData-VocCap]**.

Example 105: a batch request that contains the following operations in the order listed:

A change set that contains the following requests:

- *Insert a new entity (with* `Content-ID` = 1*)*
- *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
Content-Length: ###

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

--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
Content-Type: application/http
Content-ID: 1

POST /service/Customers HTTP/1.1
Host: host
Content-Type: application/json
Content-Length: ###

<JSON representation of a new Customer entity>
--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
Content-Type: application/http
Content-ID: 2

POST $1/Orders HTTP/1.1
Host: host
```

```
Content-Type: application/json
Content-Length: ###

<JSON representation of a new Order>
--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd--
--batch_36522ad7-fc75-4b56-8c71-56071383e77b--
```

The response contains relative `Location` *headers.*

```
HTTP/1.1 200 OK
OData-Version: 4.0
Content-Length: ####
Content-Type: multipart/mixed; boundary=batch_36522ad7-fc75-4b56-8c71-56071383e77a

--batch_36522ad7-fc75-4b56-8c71-56071383e77a
Content-Type: multipart/mixed; boundary=changeset_77162fcd-b8da-41ac-a9f8-9357efbbe

--changeset_77162fcd-b8da-41ac-a9f8-9357efbbe
Content-Type: application/http

HTTP/1.1 201 OK
Location: Customers('ALFKI')
…
--changeset_77162fcd-b8da-41ac-a9f8-9357efbbe
Content-Type: application/http

HTTP/1.1 201 OK
Location: Orders(1)
…
--changeset_77162fcd-b8da-41ac-a9f8-9357efbbe--
--batch_36522ad7-fc75-4b56-8c71-56071383e77a--
```

The second `Location` *URL* `Orders(1)` *is relative with its base URI being the second request URL* `$1/Orders`*. To get an absolute base URI, the client must replace the* `$1` *with the first* `Location` *URL* `Customers('ALFKI')` *and resolve the resulting URL* `Customers('ALFKI')/Orders(1)` *relative to its base URI, which is* `http://host/service/Customers` *(determined from the first request URL* `/service/Customers` *and the* `Host: host` *header as in example 102). This gives the effective second request URL* `http://host/service/Customers('ALFKI')/Orders` *as base URI for the second* `Location` *URL, which therefore resolves to* `http://host/service/Customers('ALFKI')/Orders(1)`*.*

11.7.7.3 Referencing an ETag

Example 106: a batch request that contains the following operations in the order listed:

- *Get an employee (with* `Content-ID` = `1`*)*
- *Update the salary only if the employee has not changed*

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

--batch_36522ad7-fc75-4b56-8c71-56071383e77b
Content-Type: application/http
Content-ID: 1

GET /service/Employees(0) HTTP/1.1
Host: host
Accept: application/json


--batch_36522ad7-fc75-4b56-8c71-56071383e77b
Content-Type: application/http
Content-ID: 2

PATCH /service/Employees(0) HTTP/1.1
```

```
Host: host
Content-Type: application/json
Content-Length: ###
If-Match: $1

{
   "Salary": 75000
}
--batch_36522ad7-fc75-4b56-8c71-56071383e77b--
```

11.7.7.4 Referencing Response Body Values

Example 107: a batch request that contains the following operations in the order listed:

- *Get an employee (with* `Content-ID` = 1*)*
- *Get all employees residing in the same building*

```
POST /service/$batch HTTP/1.1
Host: host
OData-Version: 4.01
Content-Type: multipart/mixed; boundary=batch_36522ad7-fc75-4b56-8c71-56071383e77b
Content-Length: ###

--batch_36522ad7-fc75-4b56-8c71-56071383e77b
Content-Type: application/http
Content-ID: 1

GET /service/Employees/0?$select=Building HTTP/1.1
Host: host
Accept: application/json


--batch_36522ad7-fc75-4b56-8c71-56071383e77b
Content-Type: application/http
Content-ID: 2

GET /service/Employees?$filter=Building eq $1/Building HTTP/1.1
Host: host
Accept: application/json


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

11.7.7.5 Processing a Multipart Batch Request

The service MUST process the individual requests and change sets within a multipart batch request in the order received. Processing stops on the first error unless the `continue-on-error` preference is specified with an explicit or implicit value of `true`.

All requests 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 apply this all-or-nothing requirement. The service MAY execute the requests within a change set in any order and MAY return the responses to the individual requests in any order. If a request specifies a request identifier, the service MUST include the `Content-ID` header with the request identifier in the corresponding response so clients can correlate requests and responses.

11.7.7.6 Multipart Batch Response

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

The body of a multipart response to a multipart batch request MUST structurally match one-to-one with the multipart batch request body, such that the same multipart message structure defined for requests is used for responses.

There are three 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, is returned that applies to all requests in the change set and MUST be a valid OData error response.
- When an error occurs processing a request and the `continue-on-error` preference is not specified, or specified with an explicit value of `false`, processing of the batch is terminated and the error response is the last part of the multipart response.
- Asynchronously processed batch requests can return interim results and end with a `202 Accepted` as the last part of the multipart response. Therefore, the `respond-async` preference MUST NOT be applied to individual requests within a batch if the batch response is a multipart response.

The body of a multipart response to a JSON batch request contains one body part for each processed or accepted request. The order of the body parts is insignificant as each body part MUST contain the `Content-ID` header with the value of the `id` name/value pair of the corresponding request object.

A response to an operation in a batch MUST be formatted exactly as it would have appeared outside of a batch as described in the corresponding subsections of chapter Data Service Requests. Relative URLs in each individual response are relative to the request URL of the corresponding individual request (see example 105). URLs in responses MUST NOT contain `$`-prefixed request identifiers.

Example 108: referencing the batch request example 104 above, assume all the requests except the final query request succeed. In this case the response would be

```
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

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: ###

<JSON representation of the Customer entity with key ALFKI>
--b_243234_25424_ef_892u748
Content-Type: multipart/mixed; boundary=cs_12u7hdkin252452345eknd_383673037

--cs_12u7hdkin252452345eknd_383673037
Content-Type: application/http
Content-ID: 1

HTTP/1.1 201 Created
Content-Type: application/json
Location: http://host/service.svc/Customer('POIUY')
Content-Length: ###

<JSON representation of the new Customer entity>
--cs_12u7hdkin252452345eknd_383673037
Content-Type: application/http
Content-ID: 2

HTTP/1.1 204 No Content


--cs_12u7hdkin252452345eknd_383673037--
--b_243234_25424_ef_892u748
Content-Type: application/http

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

```
Content-Length: ###

<Error message>
--b_243234_25424_ef_892u748---
```

11.7.7.7 Asynchronous Batch Requests

Batch requests MAY be executed asynchronously by including the `respond-async` preference in the `Prefer` header. If the service responds with a multipart batch response, it MUST ignore the `respond-async` preference for individual requests within a batch.

After successful execution of the batch request the response to the batch request is returned in the body of a response to an interrogation request against the *status monitor resource* URL (see Asynchronous Requests).

A service MAY return interim results to an asynchronously executing batch. It does this by responding with `200 OK` to a `GET` request to the monitor resource and including a `202 Accepted` response as the last part of the multipart response. The client can use the monitor URL returned in this `202 Accepted` response to continue processing the batch response.

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

Example 109: referencing the example 104 above again, assume that

```
HTTP/1.1 202 Accepted
Location: http://service-root/async-monitor-0
Retry-After: ###
```

When interrogating the monitor URL only the first request in the batch has finished processing and all the remaining requests are still being processed. Note that the actual multipart batch response itself is contained in an `application/http` *wrapper as it is a response to a status monitor resource:*

```
HTTP/1.1 200 OK
Content-Type: application/http

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

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: ###

<JSON representation of the Customer entity with key ALFKI>
--b_243234_25424_ef_892u748
Content-Type: application/http

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


--b_243234_25424_ef_892u748--
```

After some time the client makes a second request using the returned monitor URL, not explicitly accepting `application/http`*. The batch is completely processed and the response is the final result.*

```
HTTP/1.1 200 OK
AsyncResult: 200
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

--cs_12u7hdkin252452345eknd_383673037
Content-Type: application/http
Content-ID: 1

HTTP/1.1 201 Created
Content-Type: application/json
Location: http://host/service.svc/Customer('POIUY')
Content-Length: ###

<JSON representation of a new Customer entity>
--cs_12u7hdkin252452345eknd_383673037
Content-Type: application/http
Content-ID: 2

HTTP/1.1 204 No Content


--cs_12u7hdkin252452345eknd_383673037--
--b_243234_25424_ef_892u748
Content-Type: application/http

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

<Error message>
--b_243234_25424_ef_892u748--
```

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 4.0 Service Conformance Levels

OData 4.0 defines three levels of conformance for an OData Service.

Note: 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 4.0 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 4.0 Intermediate Conformance Level. Services that conform to the OData 4.0 Advanced Conformance Level can expect to interoperate with the most functionality against the broadest range of generic clients.

Services can advertise their level of conformance by annotating their entity container with the term `Capabilities.ConformanceLevel` defined in **[OData-VocCap]**.

Note: 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 4.0 Minimal Conformance Level

In order to conform to the OData 4.0 Minimal conformance level, a service:

1. MUST publish a service document at the service root (section 11.1.1)
2. MUST return data according to the **[OData-JSON]** format
3. MUST use server-driven paging when returning partial results (section 11.2.6.7) and not use any other mechanism
4. MUST return the appropriate `OData-Version` header (section 8.1.5)
5. MUST conform to the semantics the following headers, or fail the request
    1. `Accept` (section 8.2.1)
    2. `OData-MaxVersion` (section 8.2.7)
6. MUST follow OData guidelines for extensibility (section 6 and all subsections)
7. MUST successfully parse the request according to **[OData-ABNF]** for any supported system query options and follow the specification or fail the request
8. MUST expose only data types defined in **[OData-CSDLXML]**
9. MUST NOT require clients to understand any metadata or instance annotations (section 6.4), custom headers (section 6.5), or custom content (section 6.2) in the payload in order to correctly consume the service
10. MUST NOT violate any OData update semantics (section 11.4 and all subsections)
11. MUST NOT violate any other OData-defined semantics
12. SHOULD support `$expand` (section 11.2.5.2)
13. SHOULD publish metadata at `$metadata` according to **[OData-CSDLXML]** and MAY publish metadata according to **[OData-CSDLJSON]** (section 11.1.2)
14. MUST support prefixed variants of supported headers and preference values
15. MUST support enumeration and duration literals in URLs with the type prefix

Additionally, if async operations are supported:

16. MUST return an HTTP message as the final response to an asynchronous request with an `OData-MaxVersion` value of `4.0` and an `Accept` header including `application/http`.
17. MAY return the `AsyncResult` header in the final response to an asynchronous request

To be considered an *Updatable OData Service*, the service additionally:

18. MUST include edit links (explicitly or implicitly) for all updatable or deletable resources according to **[OData-JSON]**
19. MUST support `POST` of new entities to insertable entity sets (section 11.4.1.5)
20. MUST support `POST` of new related entities to updatable navigation properties (section 11.4.2)
21. MUST support `POST` to `$ref` to add an existing entity to an updatable related collection (section 11.4.6.1)
22. MUST support `PUT` to `$ref` to set an existing single updatable related entity (section 11.4.6.3)
23. MUST support `PATCH` to all edit URLs for updatable resources (section 11.4.3)
24. MUST support `DELETE` to all edit URLs for deletable resources (section 11.4.5)
25. MUST support `DELETE` to `$ref` to remove a reference to an entity from an updatable navigation property (section 11.4.6.2)
26. MUST support `If-Match` header in update/delete of any resources returned with an ETag (section 11.4.1.1)
27. MUST return a `Location` header with the edit URL or read URL of a created resource (section 11.4.2)
28. MUST include the `OData-EntityId` header in response to any create or upsert operation that returns `204 No Content` (section 8.3.4)
29. MUST support Upserts (section 11.4.4)
30. SHOULD support `PUT` and `PATCH` to an individual primitive (section 11.4.9.1) or complex (section 11.4.9.3) property (respectively)
31. SHOULD support `DELETE` to set an individual property to null (section 11.4.9.2)
32. SHOULD support deep inserts (section 11.4.2.2)
33. MAY support set-based updates (section 11.4.13) or deletes (section 11.4.14) to members of a collection

12.1.2 OData 4.0 Intermediate Conformance Level

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

1. MUST conform to the OData 4.0 Minimal Conformance Level
2. MUST successfully parse the request according to **[OData-ABNF]** and follow the specification or fail the request
3. MUST support `$select` (section 11.2.5.1)
4. MUST support casting to a derived type according to **[OData-URL]** if derived types are present in the model
5. MUST support `$top` (section 11.2.6.3)
6. MUST support `/$value` on media entities (section 11.1.2) and individual properties (section 11.2.4.2)
7. MUST support `$filter` (section 11.2.6.1)
    1. MUST support `eq`, `ne` filter operations on properties of entities in the requested entity set (section 11.2.6.1.1)
    2. MUST support aliases in `$filter` expressions (section 11.2.6.1.3)
    3. SHOULD support additional filter operations (section 11.2.6.1.1) and MUST fail the request for any unsupported filter operations
    4. SHOULD support the canonical functions (section 11.2.6.1.2) and MUST fail the request for any unsupported canonical functions
    5. SHOULD support `$filter` on expanded entities (section 11.2.5.2.1)
8. SHOULD publish metadata at `$metadata` according to **[OData-CSDLXML]** (section 11.1.2)
9. SHOULD support the **[OData-JSON]** format

10. SHOULD consider supporting basic authentication as defined in **[RFC7617]** over HTTPS for the highest level of interoperability with generic clients
11. SHOULD support the `$search` system query option (section 11.2.6.6)
12. SHOULD support the `$skip` system query option (section 11.2.6.4)
13. SHOULD support the `$count` system query option (section 11.2.6.5)
14. SHOULD support `$expand` (section 11.2.5.2)
15. SHOULD support the lambda operators `any` and `all` on navigation- and collection-valued properties (section 5.1.1.10 in **[OData-URL]**)
16. SHOULD support the `/$count` segment on navigation and collection properties (section 11.2.10)
17. SHOULD support `$orderby` `asc` and `desc` on individual properties (section 11.2.6.2)

12.1.3 OData 4.0 Advanced Conformance Level

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

1. MUST conform to at least the OData 4.0 Intermediate Conformance Level
2. MUST publish metadata at `$metadata` according to **[OData-CSDLXML]** (section 11.1.2)
3. MUST support the **[OData-JSON]** format
4. MUST support the `/$count` segment on navigation and collection properties (section 11.2.10)
5. MUST support the lambda operators `any` and `all` on navigation- and collection-valued properties (section 5.1.1.10 in **[OData-URL]**)
6. MUST support the `$skip` system query option (section 11.2.6.4)
7. MUST support the `$count` system query option (section 11.2.6.5)
8. MUST support `$orderby` with `asc` and `desc` on individual properties (section 11.2.6.2)
9. MUST support `$expand` (section 11.2.5.2)
    1. MUST support returning references for expanded properties
    2. MUST support `$filter` on expanded collection-valued properties
    3. MUST support cast segment in expand with derived types
    4. SHOULD support `$orderby` with `asc` and `desc` on expanded collection-valued properties
    5. SHOULD support `$count` on expanded collection-valued properties
    6. SHOULD support `$top` and `$skip` on expanded collection-valued properties
    7. SHOULD support `$search` on expanded collection-valued properties
    8. SHOULD support `$levels` for recursive expand (section 11.2.5.2.1.1)
    9. MAY support `$compute` on expanded properties
10. MUST support the `$search` system query option (section 11.2.6.6)
11. MUST support batch requests according to the multipart format (section 11.7 and all subsections) and MAY support batch requests according to the JSON Batch format defined in **[OData-JSON]**
12. MUST support the resource path conventions defined in **[OData-URL]**
13. SHOULD support asynchronous requests (section 11.6)
14. SHOULD support Delta change tracking (section 11.3)
15. SHOULD support cross-join queries defined in **[OData-URL]**
16. MAY support the `$compute` system query option (section 11.2.5.3)

12.2 OData 4.01 Service Conformance Levels

OData services can report conformance to the OData 4.01 specification by including `4.01` in the list of supported protocol versions in the `Core.ODataVersions` annotation, as defined in **[OData-VocCore]**. As all OData 4.01 compliant services must also be fully OData 4.0 compliant, OData 4.01 services do not need to separately list `4.0` as a supported version.

12.2.1 OData 4.01 Minimal Conformance Level

In order to conform to the OData 4.01 Minimal Conformance Level, a service:

1. MUST conform to the OData 4.0 Minimal Conformance Level
2. MUST be compliant with version 4.01 of the **[OData-JSON]** format
3. MUST return the `AsyncResult` result header in the final response to an asynchronous request if asynchronous operations are supported.
4. MUST support both prefixed and non-prefixed variants of supported headers and preference values
5. MUST reject a request with an incompatible `$schemaversion` system query option if a `Core.SchemaVersion` annotation is returned in `$metadata`
6. MUST support specifying supported system query options with or without the `$` prefix
7. MUST support case-insensitive query option, operator, and canonical function names
8. MUST return identifiers in the case they are specified in `$metadata`
9. MUST support both 4.0 and 4.01 syntax in URLs for supported functionality regardless of requested `OData-MaxVersion`
    1. MUST support casting strings to primitive types in URLs
    2. MUST support enumeration and duration literals in URLs with or without the type prefix
    3. MUST support invoking parameter-less function imports with or without parentheses
    4. MUST support an empty object or no-content for the request body when invoking an action with no non-binding parameters
    5. MUST support invoking functions and actions in a default namespace with or without namespace qualification
    6. MUST support parameter aliases for key values and function parameter values if they allow the octets `00` (NUL), `2F` (forward slash), or `5C` (backslash) in string literals
    7. SHOULD support implicit aliasing of parameters
    8. SHOULD support `eq`/`ne` `null` comparison for navigation properties with a maximum cardinality of one
    9. SHOULD support the `in` operator
    10. SHOULD support `divby`
    11. SHOULD support negative indexes for the substring function
    12. MAY support Key-As-Segment URL convention
        1. MUST also support canonical URL conventions (described in **[OData-URL]**) or include URLs in payload
    13. MAY support the count of a filtered collection in a common expression
    14. MAY support equal and non-equal structural comparison
10. SHOULD publish metadata at `$metadata` according to both **[OData-CSDLXML]** and **[OData-CSDLJSON]** (section 11.1.2)
11. SHOULD NOT have identifiers within a uniqueness scope (e.g. a schema, a structural type, or an entity container) that differ only by case
12. SHOULD return the `Core.ODataVersions` annotation
13. SHOULD report capabilities through the Capabilities vocabulary
14. MAY support filtering on annotation values
15. MAY support `$compute` system query option
16. MAY support `$search` for all collections
17. MAY support 4.01 behavior, including returning 4.01 content and payloads, if the client does not specify the `OData-MaxVersion:4.0` request header

In addition, to be considered an *Updatable OData 4.01 Service*, the service:

18. MUST conform to the OData 4.0 Minimal Conformance Level for an Updateable service.

19. MUST support `DELETE` to the reference of a collection member to be removed, identified by key (section 11.4.6.2)
20. SHOULD support `PUT` against single entity with nested content
21. SHOULD support deep updates (section 11.4.3.1) and deep inserts (section 11.4.2.2)
22. SHOULD support `PUT` or `DELETE` to `$ref` of a collection-valued nav prop
23. MAY support `POST` to collections of complex/primitive types
24. MAY support `PATCH` and `DELETE` to a collection
25. MAY support `POST`, `PATCH` and `DELETE` to a collection URL terminating in a type cast segment
26. MAY support `PATCH` to entity sets using the 4.01 delta payload format
27. MAY support `$select` and `$expand` on data modification requests

12.2.2 OData 4.01 Intermediate Conformance Level

In order to conform to the OData 4.01 Intermediate Conformance Level, a service:

1. MUST conform to the OData 4.01 Minimal Conformance Level
2. MUST conform to the OData 4.0 Intermediate Conformance Level
3. MUST support `eq`/`ne` `null` comparison for navigation properties with a maximum cardinality of one
4. MUST support the `in` operator
5. MUST support the `$select` option nested within `$select`
6. SHOULD support the count of a filtered collection in a common expression
7. SHOULD support equal and non-equal structural comparison
8. SHOULD support `$compute` system query option
9. SHOULD support nested query options in `$select`
10. MAY support nested parameter alias assignments in `$select` and `$expand`
11. MAY support filtering a collection using a `/$filter` path segment

12.2.3 OData 4.01 Advanced Conformance Level

In order to conform to the OData 4.01 Advanced Conformance Level, a service:

1. MUST conform to the OData 4.01 Intermediate Conformance Level
2. MUST conform to the OData 4.0 Advanced Conformance Level
3. MUST support the count of a filtered/searched collection in a common expression
4. MUST support `$compute` system query option
5. MUST support nested options in `$select`
    1. MUST support `$filter` on selected collection-valued properties
    2. SHOULD support `$orderby` with `asc` and `desc` on selected collection-valued properties
    3. SHOULD support the `$count` on selected collection-valued properties
    4. SHOULD support `$top` and `$skip` on selected collection-valued properties
    5. SHOULD support `$search` on selected collection-valued properties
6. MUST publish metadata at `$metadata` according to **[OData-CSDLJSON]** (section 11.1.2)
7. MUST support batch requests according both to the multipart format (section 11.7 and all subsections) and the JSON Batch format defined in **[OData-JSON]**
8. SHOULD support filtering a collection using a `/$filter` path segment
9. SHOULD support nested parameter alias assignments in `$select` and `$expand`
10. MAY support case-insensitive comparison of identifiers in URLs and request payloads if no exact match is found, using the same lookup sequence as for default namespaces with a case-insensitive comparison

12.3 Interoperable OData Clients

Interoperable OData clients can expect to work with OData Services that comply with at least the OData 4.0 Minimal Conformance Level and implement the **[OData-JSON]** format.

To be generally interoperable, OData clients

1. MUST specify the `OData-MaxVersion` header in requests (section 8.2.7)
2. MUST specify `OData-Version` (section 8.1.5) and `Content-Type` (section 8.1.1) in any request with a payload
3. MUST be a conforming consumer of OData as defined in **[OData-JSON]**
4. MUST follow redirects (section 9.1.5)
5. MUST correctly handle next links (section 11.2.6.7)
6. MUST support instances returning properties and navigation properties not specified in metadata (section 11.2)
7. MUST generate `PATCH` requests for updates, if the client supports updates (section 11.4.3)
8. MUST include the `$` prefix when specifying OData-defined system query options
9. MUST use case-sensitive query options, operators, and canonical functions
10. MUST encode the plus character (octet `0x2B`) as `%2B` in URLs to avoid servers mis-interpreting the plus character as an encoded space
11. SHOULD support basic authentication as defined in **[RFC7617]** over HTTPS
12. MAY request entity references in place of entities previously returned in the response (section 11.2.8)
13. MAY support deleted entities, link entities, deleted link entities in a delta response (section 11.3)
14. MAY support asynchronous responses (section 11.6)
15. MAY support `metadata=minimal` in a JSON response (see **[OData-JSON]**)
16. MAY support `streaming` in a JSON response (see **[OData-JSON]**)

In addition, interoperable OData 4.01 clients

17. MUST send OData 4.0-compliant payloads to services that don't advertise support for 4.01 or greater through the `Core.ODataVersions` metadata annotation (see **[OData-VocCore]**)
18. MUST specify identifiers in payloads and URLs in the case they are specified in `$metadata`
19. MUST be prepared to receive any valid 4.01 CSDL
20. MUST be prepared to receive any valid 4.01 response according to the requested format
21. SHOULD use capabilities (see **[OData-VocCap]**) to determine if a 4.01 feature is supported but MAY attempt syntax and be prepared to handle `400 Bad Request` or `501 Not Implemented`

Appendix A. References

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

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

A.1 Normative References

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

[OData-ABNF]

ABNF components: OData ABNF Construction Rules Version 4.02 and OData ABNF Test Cases.
See link in "Related work" section on cover page.

[OData-Aggregation]

OData Extension for Data Aggregation Version 4.02.
See link in "Related work" section on cover page.

[OData-CSDL]

OData Common Schema Definition Language (CSDL) JSON Representation Version 4.02.
See link in "Related work" section on cover page.

OData Common Schema Definition Language (CSDL) XML Representation Version 4.02.
See link in "Related work" section on cover page.

[OData-JSON]

OData JSON Format Version 4.02.
See link in "Related work" section on cover page.

[OData-URL]

OData Version 4.02. Part 2: URL Conventions.
See link in "Related work" section on cover page.

[OData-VocCap]

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

[OData-VocCore]

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

[RFC2046]

Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996. https://www.rfc-editor.org/info/rfc2046.

[RFC2119]

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

[RFC3987]

Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, DOI 10.17487/RFC3987, January 2005. https://www.rfc-editor.org/info/rfc3987.

[RFC5646]

Phillips, A., Ed., and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009. https://www.rfc-editor.org/info/rfc5646.

[RFC5789]

Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, DOI 10.17487/RFC5789, March 2010. https://www.rfc-editor.org/info/rfc5789.

[RFC7240]

Snell, J., "Prefer Header for HTTP", RFC 7240, DOI 10.17487/RFC7240, June 2014. https://www.rfc-editor.org/info/rfc7240.

[RFC7617]

Reschke, J., "The 'Basic' HTTP Authentication Scheme", RFC 7617, DOI 10.17487/RFC7617, September 2015. https://www.rfc-editor.org/info/rfc7617.

[RFC8174]

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

[RFC9110]

Fielding, R., Ed., M. Nottingham, Ed., and J. Reschke, Ed., "HTTP Semantics", RFC 9110, June 2022 https://www.rfc-editor.org/info/rfc9110.

A.2 Informative References

[ECMAScript]

ECMAScript 2023 Language Specification, 14th Edition, June 2023. Standard ECMA-262. https://www.ecma-international.org/publications-and-standards/standards/ecma-262/.

[Well-Known Text]

OpenGIS Implementation Specification for Geographic information – Simple feature access – Part 1: Common architecture, May 2011. Open Geospatial Consortium. https://www.ogc.org/standard/sfa/.

Appendix B. Safety, Security and Privacy Considerations

This section is provided as a service to the application developers, information providers, and users of OData version 4.0 giving some references to starting points for securing OData services as specified. OData is a REST-full multi-format service that depends on other services and thus inherits both sides of the coin, security enhancements and concerns alike from the latter.

For HTTP relevant security implications please cf. the relevant sections of **[RFC9110]** (17. Security Considerations) and for the HTTP `PATCH` method **[RFC5789]** (5. Security Considerations) as starting points.

B.1 Authentication

OData Services requiring authentication SHOULD consider supporting basic authentication as defined in **[RFC7617]** over HTTPS for the highest level of interoperability with generic clients. They MAY support other authentication methods.

Appendix C. Acknowledgments

C.1 Special Thanks

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

- Howard Abrams (CA Technologies)
- Ken Baclawski (Northeastern University)
- Jay Balunas (Red Hat)
- Stephen Berard (Schneider Electric Industries SAS)
- Mark Biamonte (Progress Software)
- Matthew Borges (SAP SE)
- 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)
- Yi Ding (Microsoft)
- Diane Downie (Citrix Systems)
- Patrick Durusau (Individual)
- Andrew Eisenberg (IBM)
- Chet Ensign (OASIS)
- Davina Erasmus (SDL)
- George Ericson (Dell)
- Colleen Evans (Microsoft)
- Jason Fam (IBM)
- Senaka Fernando (WSO2)
- Josh Gavant (Microsoft)
- Brent Gross (IBM)
- Zhun Guo (Individual)
- Anila Kumar GVN (CA Technologies)
- Stefan Hagen (Individual)
- Ralf Handl (SAP SE)
- Barbara Hartel (SAP SE)
- Hubert Heijkers (IBM)
- Jens Hüsken (SAP SE)
- Evan Ireland (SAP SE)
- Gershon Janssen (Individual)
- Ram Jeyaraman (Microsoft)
- Ling Jin (IBM)
- Ted Jones (Red Hat)
- Diane Jordan (IBM)
- Stephan Klevenz (SAP SE)
- Gerald Krause (SAP SE)
- 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 SE)
- Nuccio Piscopo (iFOSSF – International Free and Open Source Solutions Foundation)
- Michael Pizzo (Microsoft)
- Ramesh Reddy (Red Hat)
- Robert Richards (Mashery)
- Sumedha Rubasinghe (WSO2)
- James Snell (IBM)
- Christof Sprenger (Microsoft)
- Heiko Theißen (SAP SE)
- Jeffrey Turpin (Axway Software)
- John Willson (Individual)
- John Wilmes (Individual)
- Christopher Woodruff (Perficient, Inc.)
- Martin Zurmuehl (SAP SE)

C.2 Participants

OData TC Members:

First Name	Last Name	Company
George	Ericson	Dell
Hubert	Heijkers	IBM
Ling	Jin	IBM
Stefan	Hagen	Individual
Michael	Pizzo	Microsoft
Christof	Sprenger	Microsoft
Ralf	Handl	SAP SE
Gerald	Krause	SAP SE
Heiko	Theißen	SAP SE

Appendix D. Revision History

Revision	Date	Editor	Changes Made
Committee Specification Draft 01	2024-02-28	Michael Pizzo Ralf Handl Heiko Theißen	Import material from OData Version 4.01 Part 1: Protocol Changes listed in section 1.1

Appendix E. 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.

As stated in the OASIS IPR Policy, the following three paragraphs in brackets apply to OASIS Standards Final Deliverable documents (Committee Specification, Candidate OASIS Standard, OASIS Standard, or Approved Errata).

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

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

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

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 https://www.oasis-open.org/policies-guidelines/trademark/ for above guidance.