What's New in OData Version 4.0

Committee Note 01

15 August 2013

Specification URIs

This version:

http://docs.oasis-open.org/odata/new-in-odata/v4.0/cn01/new-in-odata-v4.0-cn01.doc (Authoritative)
http://docs.oasis-open.org/odata/new-in-odata/v4.0/cn01/new-in-odata-v4.0-cn01.html
http://docs.oasis-open.org/odata/new-in-odata/v4.0/cn01/new-in-odata-v4.0-cn01.pdf

Previous version:

N/A

Latest version:

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

Technical Committee:

OASIS Open Data Protocol (OData) TC

Chairs:

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

Editors:

Mike Pizzo (mikep@microsoft.com), Microsoft
Ralf Handl (ralf.handl@sap.com), SAP AG
Stefan Drees (stefan@drees.name), Individual
Martin Zurmuehl (martin.zurmuehl@sap.com), SAP AG

Related work:

This document is related to:

- *OData Version 4.0 Part 1: Protocol*. Latest version. http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part1-protocol.html

This is a Non-Standards Track Work Product. The patent provisions of the OASIS IPR Policy do not apply.

- *OData Version 4.0 Part 2: URL Conventions*. Latest version. http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part2-url-conventions.html
- *OData Version 4.0 Part 3: Common Schema Definition Language (CSDL)*. Latest version. http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part3-csdl.html
- *OData Atom Format 4.0.* Latest version. http://docs.oasis-open.org/odata/odata-atom-format/v4.0/odata-atom-format-v4.0.html
- *OData JSON Format 4.0.* Latest version. http://docs.oasis-open.org/odata/odata-json-format/v4.0/odata-json-format-v4.0.html

Abstract:

This document describes how OData Version 4.0 differs from its predecessor version, and why.

Status:

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

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

Citation format:

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

[New-in-OData]

What’s New in OData Version 4.0. 15 August 2013. OASIS Committee Note 01. http://docs.oasis-open.org/odata/new-in-odata/v4.0/cn01/new-in-odata-v4.0-cn01.html.

Copyright © OASIS Open 2013. All Rights Reserved.

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.

This is a Non-Standards Track Work Product.
The patent provisions of the OASIS IPR Policy do not apply.

Table of Contents

This is a Non-Standards Track Work Product.
The patent provisions of the OASIS IPR Policy do not apply.

This is a Non-Standards Track Work Product.
The patent provisions of the OASIS IPR Policy do not apply.

This is a Non-Standards Track Work Product.
The patent provisions of the OASIS IPR Policy do not apply.

This is a Non-Standards Track Work Product.
The patent provisions of the OASIS IPR Policy do not apply.

1 Introduction

The Open Data Protocol (OData) was initially proposed by Microsoft and developed through an open process, incorporating features, requirements, and feedback from the community. OData was developed through version 3.0 before being contributed to the OASIS OData Technical committee in July of 2012. This document describes the changes made to Microsoft's OData Version 3.0 that resulted in OASIS OData Version 4.0 and the rationale behind these changes.

In evolving a specification over time, sometimes you find things that worked out better than you had expected and other times you find there are things you wish you had done differently. Although most of OData landed squarely in the positive camp, there were some things we realized, with 20-20 hindsight, could be improved. In producing OData Version 4.0, in addition to adding incremental features to address common requirements, the committee took breaking changes where required to clean up some of the legacy introduced through the trials, errors, and lessons learned from earlier versions of the specification. These changes were inspired by a small set of key desires:

- Move from "structured data on the web" to a "web of structured data", tearing down the boundaries between formerly isolated OData services.
- Allow requesting exactly the desired subset of data with as few roundtrips as possible, and still keep the query language simple and intuitive.
- Make all features of OData combine well with each other, keeping each single feature as simple as possible and avoiding several ways to achieve the same goal.

This document follows the structure of the specification documents and cross-references related changes. It is non-normative, so the key words "MUST", "SHOULD", "MAY", and "NOT" are avoided and readers MUST NOT assume it states any requirements.

1.1 References (non-normative)

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

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

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

[OData-Protocol] *OData Version 4.0 Part 1: Protocol.*
See link in "Related work" section on cover page.

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

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989. http://www.ietf.org/rfc/rfc1122.txt.

This is a Non-Standards Track Work Product.
The patent provisions of the OASIS IPR Policy do not apply.

2 OData Part 1: Protocol

2.1 Versioning

2.1.1 Improved: Protocol Versioning

Services now respond with the maximum protocol version supported by the server and indicated acceptable by the client.

Also "downgrade" to versions prior to 4.0 is not covered, and service publishers are advised to use new service root URLs for 4.0 services.

2.1.2 New: Model Versioning

V4 now defines which changes to a model are considered "non-breaking". Services are allowed to reuse the same root URL as long as their model changes "non-breaking".

On the other hand clients are now required to tolerate these "non-breaking" changes, and services are required to switch to a different root URL when introducing "breaking" changes to their model.

The set of "non-breaking" changes is tailored to reduce the burden on clients; it's an application of the Robustness Principle (a.k.a. Postel's law) cf. **[RFC1122]**: "Be liberal in what you accept and conservative in what you send". An OData client is expected to "silently ignore" instead of "complain about" any "unknown stuff" present in a response and still make the most of it.

2.2 Extensibility

2.2.1 New: Function Extensibility for `$filter` and `$orderby`

Custom functions defined with the new CSDL element `Function` may be used in `$filter` and `$orderby` expressions.

2.2.2 Improved: Vocabulary Extensibility

Vocabulary-based annotations have become more powerful, and instance data can now be annotated in both Atom and JSON in addition to annotating metadata.

2.2.3 Pruned: Metadata Extensibility via Arbitrary Elements and Attributes

Metadata can now be annotated solely with vocabulary-based annotations. The extension points for arbitrary XML elements (`xs:any`) or attributes (`xs:anyAttribute`) are gone.

This is a Non-Standards Track Work Product.
The patent provisions of the OASIS IPR Policy do not apply.

2.3 Header Fields

2.3.1 New: OData-Isolation

Snapshot isolation guarantees that a request is processed without side-effects from other requests that the service is processing at the same time. This can be interesting for batch request as well as in combination with server-driven paging, in which case it guarantees that consecutive "pages" accessed via next links won't include changes that occurred after the initial request was processed.

2.3.2 New: Asynchronous Processing and Notification Callback

Asynchronous processing can be requested by the standard preference `respond-async`. The response will point to a monitor resource that can be polled for the processing status and final result. If that's too tedious, a callback URL can be provided with the `odata.callback` preference, and the service will ping when processing is complete.

2.3.3 Improved: `Prefer` Header

OData now heavily uses the `Prefer` header and clients can express their preference for

- `odata.allow-entityreferences` for returning entity references instead of repeated entities,
- `odata.callback` for notifications for asynchronous results and delta results.
- `odata.continue-on-error` for
- `odata.include-annotations` for inclusion of annotations in responses,
- `odata.maxpagesize` for the maximum page size for server-driven paging,
- `odata.track-changes` for delta support,
- `return` for returning content or not, and
- `respond-async` for asynchronous request processing, and

2.3.4 Improved: Header names start with `OData`

- `OData-Version` replaced the former `DataServiceVersion`,
- `OData-MaxVersion` replaced `MaxDataServiceVersion`, and
- `OData-EntityId` replaced `DataServiceId`.

2.3.5 Pruned: `MinDataServiceVersion`

Services pick the maximum protocol version supported and indicated acceptable by the client, so there's no need for a lower boundary any more.

This is a Non-Standards Track Work Product.
The patent provisions of the OASIS IPR Policy do not apply.

2.4 Common Response Status Codes

2.4.1 New: `202 Accepted`

OData now defines a call choreography for asynchronous processing of requests. Clients that explicitly request asynchronous processing can be redirected to a monitor resource that tells them whether the request is still being processed, or whether it was completed, and how.

2.4.2 New: `3xx Redirection`

Services are now explicitly allowed to redirect clients to a different location, and clients are obliged to follow redirection responses.

2.4.3 New: `501 Not Implemented`

Services that haven't implemented a requested feature now have to respond with `501 Not Implemented`.

2.5 New: Context URL and Metadata ETag

The context URL describes the content of the payload, and provides a link to the corresponding "section" of the metadata document. This is used in both Atom and JSON.

The context URL can be accompanied by a metadata ETag, so clients can check whether the metadata they know matches the metadata version used for creating that response.

2.6 Requesting Data

2.6.1 New: Metadata Service

In addition to the monolithic XML metadata document services now can provide metadata as an OData service with root URL `~/$metadata/` with a fixed structure.

2.6.2 Improved: `$count`

`$inlinecount=allpages` has been shortened to `$count=true`.

And the suffix `/$count` can be appended to path segments identifying a collection, in the resource path as well as in `$filter` and `$orderby` expressions.

2.6.3 New: `$search`

Free-text search across a collection of entities is now possible with the new system query option `$search`. It uses syntax common to most search engines. It can be used together with

2.6.4 Improved: `$expand`

Expanded entities can now be filtered, ordered, paged, projected, cast, … , and of course further expanded through a more composable, nested syntax.

This is a Non-Standards Track Work Product.
The patent provisions of the OASIS IPR Policy do not apply.

2.6.5 New: `$levels`

Recursive relationships can now be expanded for a specified number of levels.

2.6.6 Improved: `$select`

`$select` now only projects the "current base set", projections of expanded entities are nested within `$expand`.

2.6.7 Improved: Null value versus Not Found

`204 No Content` is consistently used to indicate a null value (for example, of a property of an existing entity or returned from a function) where `404 Not Found` is used for things that don't exist (like an unmatched key value).

2.6.8 New: Entity References

Entity references are short representations that basically consist of a link to an entity. They can appear in almost all places instead of the full-blown entity, making requests and responses much shorter.

2.6.9 New: Change Tracking

Clients can request change-tracking support when querying a collection with the new `odata.track-changes` preference. If the server supports it, it simply includes a "delta link" in the response. Following this link a client can get the list of changes that happened since the delta link was issued, plus a new delta link to continue the game indefinitely. This covers change-tracking on original requests with `$expand`: changes to related entities as well as changes of relationships between entities are part of the delta response.

2.7 Data Modification

2.7.1 Improved: Use `PATCH` instead of `PUT`

Clients are advised to use `PATCH` instead of `PUT` because that makes it easier for them to deal with the non-breaking changes allowed to services. It also reduces wire traffic.

2.7.2 Pruned: `MERGE`

`MERGE` was used to do `PATCH` before `PATCH` existed. Now that we have `PATCH`, we no longer need `MERGE`.

2.7.3 Improved: `PATCH` with Complex Types

A `PATCH` request specifying properties of a complex type behaves the same as a `PATCH` request to an entity type; only specified properties are modified.

2.7.4 New: Upsert

`PATCH` (and `PUT`) can now create entities if the client is able to construct a valid URL for the new entity and provides all necessary data in the request.

2.8 Operations

2.8.1 New: Actions and Functions

Actions and functions are now declared on the schema level, allowing functions to be used in `$filter` expressions. `ActionImport` and `FunctionImport` now "import" the action or function declaration into a service.

2.8.2 New: Action Parameters as Atom

Action parameters may be passed in OData's Atom format to services that support Atom.

2.9 Batch

2.9.1 Improved: Error Handling

By default batch request processing now stops when encountering the first error. Clients can ask services to continue processing with the preference `odata.continue-on-error`.

2.9.2 Improved: `Content-ID`

Each request within a change set now needs a `Content-ID` header, and services will include it in the response, so clients are able to correlate requests and responses.

2.9.3 Improved: Updates outside of Change Sets

Individual data modification statements can now be members of a batch without being wrapped in a change set.

2.9.4 New: Asynchronous Batch Requests

Clients can now request that a batch request is processed asynchronously.

2.10 New: Conformance Levels

OData now defines three levels of conformance for services, and one for clients. This should make interoperability a lot easier.

This is a Non-Standards Track Work Product.
The patent provisions of the OASIS IPR Policy do not apply.

3 OData Part 2: URL Conventions

3.1 Resource Path

3.1.1 New: No Redundant Keys for Dependent Entities

Paths leading from a “parent” entity to a “child” entity don’t have to repeat key properties that are determined by the parent due to a referential constraint. If each dependent level only adds a single new key property, this results in really nice URLs, e.g. `~/Orders(1)/Items(2)`.

3.1.2 New: Addressing References via `/$ref`

Appending `/$ref` to any path that identifies an entity or a collection of entities now addresses the reference to this entity or the collection of references to these entities.

3.1.3 New: Resolving References via `~/$entity?$id=`…

A reference consists of the entity id, which may or may not be a URL that returns the entity. A new standard resource `~/$entity` allows resolving entity references. The entity id is passed in using the new system query option $id and returns a current representation of the entity.

3.1.4 Pruned: `/$links/`

The old syntax of inserting a `/$links/` segment between an entity and one of its navigation properties has been dropped as the more flexible approach of appending `/$ref` can also applied here.

3.1.5 New: Cross-Join Queries via `~/$crossjoin(`…`)`

The virtual resource `~/$crossjoin` takes a list of entity sets as its parameters and represents the Cartesian products of all entities in these sets. This works similar to SQL's `FROM` clause followed by a list of tables and views, and the Cartesian product can then be restricted to tuples matching a filter that relates properties of entities from the participating sets.

3.1.6 New: Virtual Collection `~/$all`

The new resource `~/$all` is the collection of all entities in a service. It can e.g. be used for free-text search across all entities in a service,.

3.2 Query Options

3.2.1 New: `/$count`

The path segment `/$count` for addressing the raw number of items in a collection can now be used in query expressions, so clients can now e.g. filter or sort by the number of related entities.

This is a Non-Standards Track Work Product.
The patent provisions of the OASIS IPR Policy do not apply.

3.2.2 New: Parameter Aliases

Parameter aliases are simple identifiers prefixed with an `@` sign. Formerly they could only be introduced in the resource path and then assigned a value by specifying them as a query option. Now they can in addition be used in query expressions to e.g. avoid stating the same literal multiple times, or deferring lengthy literals to a place where they are easier to read.

3.2.3 New: `$root`

The `$root` variable can be used to refer to an arbitrary entity in the same service by appending its relative canonical URL.

3.2.4 Pruned: `KEY(…)`

The `KEY(…)` syntax allowed referring to an entity in the same service by its entity set and key. This and more is now possible with `$root`.

3.2.5 Improved: `$it`

The `$it` variable can now be used everywhere in expressions to refer to the current item of the collection identified by the resource path of the request. Formerly it was only available in expressions within `any()` and `all()`.

3.2.6 New: `fractionalseconds`, `now`, `mindatetime`, `maxdatetime`

These four functions complete the set of timestamp processing functions.

3.2.7 Improved: `contains`

You now check whether a long string contains a short string with the `contains` function.

This replaces the `substringof` function which took the long string as its *second* argument; which was grammatically correct in itself and really confusing in combination with the sibling function `indexof`.

3.2.8 Improved: `totaloffsetminutes`

The tapeworm function `gettotaloffsetminutes` has lost its first syllable. Now it fits in more nicely with its siblings.

3.2.9 Pruned: `years`, `months`, `days`, `minutes`, `seconds`

These functions operated on `Edm.Time` values and have been removed together with that primitive type. Requiescat in pace.

3.2.10 Pruned: URI Literal suffixes for numeric types

Decimal, Double, Single, and Int64 literals in URIs no longer need to specify "M", "D", "F", or "L" suffixes when used in URIs.

This is a Non-Standards Track Work Product.
The patent provisions of the OASIS IPR Policy do not apply.

3.2.11 Pruned: Function Parameters as Query Options

Instead of allowing the names of function parameters as query options, clients can now introduce an alias as the parameter value in the resource path and assign a value for the alias in the query part. This is more flexible, and made the old-style syntax obsolete.

This is a Non-Standards Track Work Product.
The patent provisions of the OASIS IPR Policy do not apply.

4 OData Part 3: Common Schema Definition Language (CSDL)

4.1 EDMX Wrapper

4.1.1 New: `edmx:Include` and `edmx:IncludeAnnotations`

The `edmx:Reference` element now has two child elements that tell which parts of the referenced document are part of the referencing service.

`edmx:Include` includes a complete schema and allows to provide an alias for it. The included schema is treated as if it had been defined (with that alias) directly in the metadata document.

The edm:`Using` element was pruned as it was no longer needed.

`edmx:IncludeAnnotations` only includes annotations that match specified combination of term namespace, target namespace, and qualifier.

4.2 Types

4.2.1 New: `Edm.Date`, `Edm.Duration`, and `Edm.TimeOfDay`

These types are exactly what they claim to be: a date (without time part), a duration (years, hours, minutes, seconds, and picoseconds), and a time of day (from exactly midnight to just before the next midnight, again down to picoseconds if desired).

4.2.2 Pruned: `Edm.DateTime`

This data type convenient for some services and painful for a lot of clients: pico-second precision that may be off by 14 hours due to a wrongly guessed time zone. `Edm.DateTimeOffset` lets services share their knowledge of the time zone with their clients.

4.2.3 Pruned: `Edm.Float`

This was just a synonym for `Edm.Single`, so no loss there.

4.2.4 Pruned: Edm.Time

There were some doubts whether this data type actually *meant* the time of day and was only *represented* as duration. Now it's gone and we have `Edm.Duration` and `Edm.TimeOfDay`, both of which are exactly what they claim to be.

4.2.5 Improved: Enumeration Types

Enumeration types could be defined in CSDL V3, but it was unclear how they are represented in Atom and JSON or used in requests. We fixed that.

This is a Non-Standards Track Work Product.
The patent provisions of the OASIS IPR Policy do not apply.

4.2.6 New: Type Definitions

Type definitions allow giving a new name to a primitive type and fixing some of its facets.

DRY – don't repeat yourself, define a type instead.

4.2.7 New: Abstract Types

The abstract types `Edm.EntityType`, `Edm.ComplexType`, `Edm.PrimitiveType`, and collections of each can be used in a data model where the concrete type is not known at design time. The actual type is specified at runtime in the payload using the context URL or the type annotation.

4.3 Structural Properties

4.3.1 Improved: `Scale`

A point of some discussion was whether `Edm.Decimal` properties are fixed-point or floating point. This has been settled: it can be both. If the `Scale` is a non-negative integer, it is a fixed-point number and `Scale="variable"` indicates a floating-point decimal with up to `Precision` significant digits, and a decimal point at any possible position.

4.3.2 Pruned: `Collation`

The `Collation` attribute allowed saying e.g., "this string is sorted like an integer". We recommend using an integer in that case.

4.3.3 Pruned: `FixedLength`

Why should a client care whether a server uses, say, 1024 characters to store the string "Hello world", padded with a large number of spaces?

4.3.4 Pruned: `ConcurrencyMode`

This was a nice idea, but somewhat inflexible, so it has been replaced with an annotation using the term `Core.OptimisticConcurrencyControl`.

4.3.5 Pruned: `StoreGeneratedPatern`

This didn't belong as a model attribute, and is replaced with an annotation using the term `Core.Computed`.

4.4 Navigation Properties

Navigation properties have undergone very significant changes that now allow e.g. cross-service references necessary for mashup scenarios. The intrinsically bidirectional associations have been replaced with intrinsically unidirectional navigations.

This is a Non-Standards Track Work Product.
The patent provisions of the OASIS IPR Policy do not apply.

4.4.1 Pruned: Association, FromRole, ToRole

Less is more, see following sections.

4.4.2 Added: `Type`, `Nullable`

The new `Type` attribute specifies the entity type of the navigation targets. This type can be defined anywhere, even in a different service whose metadata is referenced and included. If the service designer only knows that the target will be OData, that's no problem: just use the abstract type `Edm.EntityType`. Multi-valued navigation is expressed via `Type="Collection(…)"`, and optional single-valued navigation via `Nullable="true"`.

4.4.3 Added: `Partner`

Bidirectional relations can be expressed with a pair of navigation properties that name each other as a `Partner`. Other than marriages, this is one-way: *A* may call *B* its partner, and *B* may or may not call *A* its partner. Maybe we should have called it `Backlink` instead.

4.4.4 Added: `OnDelete`

The “leading” entity states what will happen to related entities if it is deleted.

4.4.5 Added: `ReferentialContraint`

An entity type can now state which of its properties are tied to which properties of a related entity type. The “foreign properties” need not be key properties of the leading type.

4.5 Entity Types

4.5.1 Improved: Abstract Entity Types

Abstract entity types may now specify no key. Somewhere along the inheritance line to a concrete entity type a key has to be defined and cannot be changed after that point. This allows expressing e.g., “a person has a name and a birth date”, even if there’s no globally unique way of identifying a person.

4.5.2 Improved: Entity Keys

Entity keys can now consist of properties that are part of a complex property. Such keys must be aliased using the new `Alias` attribute of a `PropertyRef` element.

4.5.3 Improved: `HasStream`

The `HasStream` attribute is now a native EDM construct and no longer defined in a separate namespace.

This is a Non-Standards Track Work Product.
The patent provisions of the OASIS IPR Policy do not apply.

4.6 Complex Types

4.6.1 Added: Navigation Properties

Complex types can now have navigation properties with all their features. An `Address` now could navigate to a `Country`, to name the most obvious example.

4.7 Actions and Functions

Actions and functions are now defined as schema children, next to entity types and complex types. Functions can be used in filter expressions, and both actions and functions can be bound to resources.

4.7.1 New: `edm:Action`

The new `edm:Action` element defines the signature of an action, i.e. some change to resources that goes beyond a simple CUD operation. This is the predecessor of a function import that is side-effecting.

4.7.2 Improved: `edm:Function`

The `edm:Function` element defines the signature of a function that can be used in filter expressions or bound to a resource. This has more similarities to a former non-side-effecting function import than to a former “model function”.

4.7.3 Pruned: `DefiningExpression`

Functions no longer state how they are implemented. If you feel the urge to do that, just define a term in one of your vocabularies and annotate the function.

4.7.4 New: `edm:ActionImport`

The new `edm:ActionImport` element defines a top-level resource in the service that can be directly accessed with `POST` requests. It refers to an action that defines its signature and replaces a function import that has side effects.

4.7.5 Improved: `edm:FunctionImport`

The `edm:FunctionImport` element defines a top-level resource in the service that can be directly accessed with `GET` requests. It refers to a function that defines its signature and corresponds to a former function import that has no side effects, and without parameter and return-type definitions.

4.7.6 Pruned: `IsSideEffecting`

Actions and functions are now defined with separate CSDL elements.

4.7.7 Pruned: `edm:Mode`

Parameters are always input. You don't need to tell us.

4.7.8 New: Nullable return type

The `edm:ReturnType` element now includes a `Nullable` attribute to tell you whether the function can return null.

4.7.9 Pruned: `edm:CollectionType`, `edm:TypeRef`

These two provided an alternate syntax for `Collection(Some.Type)` , so we killed them. `TypeRef` allowed specifying facets for primitive types, for which we now have type definitions. It also allowed defining collections of collections of something. Just define a complex type with a collection property and squeeze it in between. This gives you a place to put additional properties that you will inevitably discover later.

4.7.10 Pruned: `edm:RowType`

This allowed defining unnamed complex types. Just define a named complex type instead.

4.7.11 Pruned: `edm:ReferenceType`

Entity references have become a very elegant, first-class concept of OData; you will not miss the reference types.

4.8 Entity Container

The good news: there is only one entity container now per service. It still has a name, but this name is rarely used, and it does not show up in URLs.

4.8.1 Pruned: `IsDefaultEntityContainer`

Unnecessary now that there is exactly one of these beasts.

4.8.2 New: `edm:Singleton`

Ever had a special entity where there can be only one of its kind? Bothered by having to define a one-element entity set for it? Use a singleton! It can be addressed directly by its name from the service root.

4.8.3 New: `IncludeInServiceDocument`

Functions that return a collection of entities behave in queries identical to an entity set. So it's natural to import parameter-less functions into the service and advertise them in the service document by setting the `IncludeInServiceDocument` attribute to `true`. Think of these as views. The same attribute can be used on entity sets to exclude them from the service document: just set it to `false`.

4.8.4 Pruned: Association Sets

Association sets were the entity-container reflection of associations, and were removed together with the associations.

4.8.5 New: Navigation Property Bindings

Navigation property bindings tell up-front into which entity set a navigation property will lead from a given start entity set. They are a lightweight, unidirectional replacement for association sets, and they are optional.

4.9 Service Document

In addition to entity sets, service documents can now expose singletons, parameterless function imports, and even nested service documents.

4.10 Vocabularies and Annotations

Vocabularies and annotations underwent a major refactoring and clean-up; now it's an extremely powerful tool, and we use it ourselves to express metadata beyond structure definitions.

4.10.1 New: Annotations instead of TypeAnnotations and ValueAnnotations

The former distinction between `TypeAnnotation` (I can cast to this “type”) and `ValueAnnotation` (this is (potentially structured) additional information) was fuzzy and proved to be a difficult choice in many scenarios. Therefore, we removed that choice and merged the two. Annotations both add additional information, and allow you to narrow (“cast”) your perception of an instance to just the annotated values.

4.10.2 New: `DefaultValue` and `AppliesTo`

Terms now can specify to which model elements they apply, and what is their default value. This allows “tagging” terms where you just apply the term and need to state only non-default values.

4.10.3 New: `edm:If`, Comparison and Logical Operators

Conditional expressions with `edm:If` no longer have to rely on existing Boolean properties, there's a whole family of operators for comparison and logical operators.

4.10.4 New: Client-Side Functions for `edm:Apply`

We define three client-side functions for `edm:Apply`: besides the obvious `odata.concat` there are `odata.fillUriTemplate` and `odata.uriEncode`. The former constructs a URL from a template, the latter encodes OData literals appropriately for URLs. Together with the new `edm:UrlRef` they allow really powerful annotations.

4.10.5 New: `edm:UrlRef`

The `edm:UrlRef` takes a string as its argument, fires a `GET` request to that string as a URL, and returns whatever is contained in the response body. Have fun!

4.10.6 Improved: `edm:AssertType` becomes `edm:Cast`

The old `edm:AssertType` expression returned its child expression as the specified type. Sounds like a Cast to us, so that's what we renamed it to.

4.10.7 New: standard vocabularies `Core`, `Measures`, `Capabilitites`, `Atom`

We use vocabularies and annotations ourselves to express metadata beyond structure information.

The `Core` vocabulary allows marking strings as URLs or language-dependent texts, properties as immutable or computed, and entity sets as ETag-protected, to name just a few. If you write your own vocabularies, you'll most likely refer to `Core`. We did in all our other vocabularies.

The `Measures` vocabulary allows marking properties as monetary amounts and measurable quantities.

The `Capabilities` vocabulary allows describing what a service can do, and what it can't, hopefully reducing the number of bad requests for clients that know how to interpret `Capabilities`.

The `Atom` vocabulary finally replaces the former `FC_TargetPath` annotation attribute.

4.10.8 Pruned: `edm:Documentation`

The former Documentation element with its two children `Summary` and `LongDescription` has been replaced by annotations `Core.Description` and `Core.LongDescription`.

4.11 New: Metadata Service

In addition to the monolithic `$metadata` XML document metadata can now be represented as an OData service with a fixed structure. This service, conveniently located at `~/$metadata/`, allows querying all types, properties, entity sets, annotations, etc. of a given OData service.

5 OData Atom Format

The Atom format has undergone only minimal changes.

5.1 Instance Metadata

5.1.1 New: Context URL and Metadata ETag

ATOM now includes a context URL in the `metadata:context` attribute of the root element of a response, and the metadata ETag in the `metadata:metadata-etag` attribute.

These attributes can also appear on a `link` element representing a navigation property.

5.2 Service Document

5.2.1 New: `metadata:service-document`

Service documents can now advertise other services that may also be of interest for the consumer. This allows e.g. mashup scenarios, where "my" service provides additional information to one or more "foreign" services. My service document can now point to the other services, just as entities of my service can navigate to entities in the other services, and my metadata document can reference the other metadata documents. A web of open data just waiting to be explored.

5.2.2 New: `metadata:singleton` and `metadata:function-import`

In addition to entity sets and related service documents singletons and parameter-less function imports can be advertised in the service document.

5.3 Improved: Primitive Types

Primitive types no longer carry the optional `Edm.` prefix. Doing things one way is easier, so in absence of any ambiguity we opted for brevity.

5.4 Improved: Individual Values

Responses that represent individual properties and action or function results that are not entities or collections of entities no longer use the property, action or function name as the name of the response root element, and instead uniformly use `metadata:value`.

5.5 New: Entity References

The element `metadata:ref` can appear instead of a repeated full `atom:entry` if preferred by the client.

This is a Non-Standards Track Work Product.
The patent provisions of the OASIS IPR Policy do not apply.

5.6 New: Delta Responses

Delta responses contain `atom:entry` elements for new and changed entities, `atom-tombstone:deleted-entry` elements for deleted entities, `metadata:link` elements for added links and `metadata:deleted-link` elements for deleted links. And of course a new delta link.

5.7 New: Action Invocation Requests

Actions are invoked with `POST` requests, and the parameters passed as properties of an implicitly defined complex type in the request body.

5.8 New: Instance Annotations

Instance annotations are represented with the new `metadata:annotation` element and allow annotation values to vary per instance.

This is a Non-Standards Track Work Product.
The patent provisions of the OASIS IPR Policy do not apply.

6 OData JSON Format

The JSON format has been completely redesigned. It is crisp and clean and has little resemblance with its predecessor, so this chapter only highlights the main features instead of listing changes. Read the full specification **[OData-JSON]**; it is the shortest document in the OData 4.0 series, and a lot of the text consists of examples.

6.1 JSON is now the Recommended Format

A service MUST implement either ATOM or JSON. We suggest JSON and require it at the highest conformance levels.

6.2 Controllable Metadata: `odata.metadata=full/minimal/none`

The amount of metadata in responses can now be controlled with the format parameter `odata.metadata`. The default is `minimal`, which is almost `none`.

Metadata is now represented as instance annotations using terms in the namespace `odata`, easily recognizable by the dot separating `odata` from the term name, e.g. `odata.context`.

6.3 Improved: Relative URLs

URLs in JSON can now be relative to the `odata.context` annotation. This annotation can appear at several levels in the payload, offering similar flexibility as the `xml:base` attribute in XML responses.

6.4 New: ContextUrl for Navigation Properties

Navigation properties that are defined outside of the model have their own `ContextUrl` to specify their metadata information.

6.5 Improved: `odata.type`

The `odata.type` annotation is now a fragment, absolute reference, or relative to its parent's `odata.type`.

Also, primitive types no longer carry the optional `Edm.` prefix. Doing things one way is easier, so in absence of any ambiguity we opted for brevity.

6.6 Improved: Serializing Numbers in JSON

`Edm.Int64` and `Edm.Decimal` values are serialized as numbers, as expected, unless the `IEEE754Compatible` format parameter is specified.

This is a Non-Standards Track Work Product.
The patent provisions of the OASIS IPR Policy do not apply.

6.7 New: Instance Annotations

Every JSON object can be annotated by adding name-value pairs whose name is the qualified name of the term (the term namespace followed by a dot and the term name), e.g. `odata.count`. And every JSON name-value pair that is not an annotation itself can be annotated by adding a name-value pair that uses the original name followed by an at sign and the qualified term name, e.g. `Orders@odata.count` to annotate the Orders navigation property.

6.8 Improved: JSON Errors

Errors in JSON now optionally include a `target` property to specify the target of an error (for example, an invalid property name) and a `details` property whose value is an array of JSON objects that can be used to report multiple errors.

6.9 New: Delta Responses, Entity References, and Whatnot

Everything defined by OData can be represented in JSON; necessarily as JSON is now the preferred format.

Appendix A. Acknowledgments

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

This is a Non-Standards Track Work Product.
The patent provisions of the OASIS IPR Policy do not apply.

Appendix B. Revision History

Revision	**Date**	**Editor**	**Changes Made**
[Rev number]	[Rev Date]	[Modified By]	[Summary of Changes]