OData Version 4.0 Part 2: URL Conventions

Committee Specification Draft ~~01~~02 / Public Review Draft ~~01~~02

~~26 April~~24 June 2013

Specification URIs

This version:

~~http://docs.oasis-open.org/odata/odata/v4.0/csprd01/part2-url-conventions/odata-v4.0-csprd01-part2-url-conventions.doc (Authoritative)~~
~~http://docs.oasis-open.org/odata/odata/v4.0/csprd01/part2-url-conventions/odata-v4.0-csprd01-part2-url-conventions.html~~
~~http://docs.oasis-open.org/odata/odata/v4.0/csprd01/part2-url-conventions/odata-v4.0-csprd01-part2-url-conventions.pdf~~
http://docs.oasis-open.org/odata/odata/v4.0/csprd02/part2-url-conventions/odata-v4.0-csprd02-part2-url-conventions.doc (Authoritative)
http://docs.oasis-open.org/odata/odata/v4.0/csprd02/part2-url-conventions/odata-v4.0-csprd02-part2-url-conventions.html
http://docs.oasis-open.org/odata/odata/v4.0/csprd02/part2-url-conventions/odata-v4.0-csprd02-part2-url-conventions.pdf

Previous version:

~~N/A~~
http://docs.oasis-open.org/odata/odata/v4.0/csprd01/part2-url-conventions/odata-v4.0-csprd01-part2-url-conventions.doc (Authoritative)
http://docs.oasis-open.org/odata/odata/v4.0/csprd01/part2-url-conventions/odata-v4.0-csprd01-part2-url-conventions.html
http://docs.oasis-open.org/odata/odata/v4.0/csprd01/part2-url-conventions/odata-v4.0-csprd01-part2-url-conventions.pdf

Latest version:

~~http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part2-url-conventions.doc (Authoritative)~~
~~http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part2-url-conventions.html~~
~~http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part2-url-conventions.pdf~~
http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part2-url-conventions.doc (Authoritative)
http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part2-url-conventions.html
http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part2-url-conventions.pdf

Technical Committee:

OASIS Open Data Protocol (OData) TC

Chairs:

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

Editors:

~~Michael Pizzo (mikep@microsoft.com), Microsoft~~

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

Additional artifacts:

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

- ~~*OData Version 4.0 Part 1: Protocol*. http://docs.oasis-open.org/odata/odata/v4.0/csprd01/part1-protocol/odata-v4.0-csprd01-part1-protocol.html~~
- *OData Version 4.0 Part 1: Protocol*. http://docs.oasis-open.org/odata/odata/v4.0/csprd02/part1-protocol/odata-v4.0-csprd02-part1-protocol.html
- *OData Version 4.0 Part 2: URL Conventions* (this document). ~~http://docs.oasis-open.org/odata/odata/v4.0/csprd01/part2-url-conventions/odata-v4.0-csprd01-part2-url-conventions.html~~http://docs.oasis-open.org/odata/odata/v4.0/csprd02/part2-url-conventions/odata-v4.0-csprd02-part2-url-conventions.html
- *OData Version 4.0 Part 3: Common Schema Definition Language (CSDL)*. ~~http://docs.oasis-open.org/odata/odata/v4.0/csprd01/part3-csdl/odata-v4.0-csprd01-part3-csdl.html~~http://docs.oasis-open.org/odata/odata/v4.0/csprd02/part3-csdl/odata-v4.0-csprd02-part3-csdl.html
- ABNF components: *OData ABNF Construction Rules Version 4.0* and *OData ABNF Test Cases.* ~~http://docs.oasis-open.org/odata/odata/v4.0/csprd01/abnf/~~http://docs.oasis-open.org/odata/odata/v4.0/csprd02/abnf/
- Vocabulary components: *OData Core Vocabulary*~~ and~~, *OData Measures Vocabulary*~~.~~ ~~http://docs.oasis-open.org/odata/odata/v4.0/csprd01/vocabularies/~~ and *OData Capabilities Vocabulary*. http://docs.oasis-open.org/odata/odata/v4.0/csprd02/vocabularies/
- XML schemas: OData EDMX XML Schema and OData EDM XML Schema. ~~http://docs.oasis-open.org/odata/odata/v4.0/csprd01/schemas/~~http://docs.oasis-open.org/odata/odata/v4.0/csprd02/schemas/
- OData Metadata Service Entity Model: ~~http://docs.oasis-open.org/odata/odata/v4.0/csprd01/models/MetadataService.edmx~~http://docs.oasis-open.org/odata/odata/v4.0/csprd02/models/MetadataService.edmx

Related work:

This specification is related to:

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

Declared XML namespaces:

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

Abstract:

The Open Data Protocol (OData) enables the creation of REST-based data services, which allow resources, identified using Uniform Resource Identifiers (URIs) and defined in an Entity Data Model (EDM), to be published and edited by Web clients using simple HTTP messages. ~~This document defines the core semantics and facilities of the protocol~~OData version 4.0 defines the core semantics and facilities of the protocol, a set of recommended (but not required) rules for constructing URLs to identify the data and metadata exposed by an OData service as well as a set of reserved URL query string operators, an Entity Data Model (EDM), and an XML representation of the entity data model exposed by an OData service. OData Atom Format version 4.0 extends the former by defining representations for OData requests and responses using an Atom format.

Status:

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

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

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

Citation format:

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

[OData-Part2]

OData Version 4.0 Part 2: URL Conventions. ~~26 April~~24 June 2013. OASIS Committee Specification Draft ~~01~~02 / Public Review Draft ~~01. http://docs.oasis-open.org/odata/odata/v4.0/csprd01/part2-url-conventions/odata-v4.0-csprd01-part2-url-conventions.html~~02. http://docs.oasis-open.org/odata/odata/v4.0/csprd02/part2-url-conventions/odata-v4.0-csprd02-part2-url-conventions.html.

Notices

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

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

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

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

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

Table of Contents

1 Introduction

The Open Data Protocol (OData) enables the creation of REST-based data services, which allow resources, identified using Uniform Resource ~~Identifiers~~Locators (URLs) and defined in a data model, to be published and edited by Web clients using simple HTTP messages. This specification defines a set of recommended (but not required) rules for constructing URLs to identify the data and metadata exposed by an OData service as well as a set of reserved URL query string operators, which if accepted by an OData service, MUST be implemented as required by this document.

The ~~[~~**[OData-Atom]**~~] and [OData-JSON]~~ and **[OData-JSON]** documents specify the format of the resource representations that are exchanged using OData and the **[OData-Protocol]** document describes the actions that can be performed on the URLs (optionally constructed following the conventions defined in this document) embedded in those representations.

Services are encouraged to follow the URL construction conventions defined in this specification when possible as consistency promotes an ecosystem of reusable client components and libraries.

1.1 Terminology

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

1.2 Normative References

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

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

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

~~[OData-JSON]~~[OData-JSON] *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 "Additional artifacts" section on cover page.

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

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

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

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax”, STD 66, RFC 3986, January 2005.
~~http://www.ietf.org/rfc/rfc3986.txt~~http://www.ietf.org/rfc/rfc3986.txt.

~~**[RFC3987]** Duerst, M. and M. Suignard, “Internationalized Resource Identifiers (IRIs)”, RFC 3987, March 1997. http://www.ietf.org/rfc/rfc3987.txt.~~

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

1.3 Typographical Conventions

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

```
Normative source code uses this paragraph style.
```

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 URL Components

A URL used by an OData service has at most three significant parts: the *service root URL*, *resource path* and *query options*. Additional URL constructs (such as a fragment) ~~MAY~~can be present in a URL used by an OData service; however, this specification applies no further meaning to such additional constructs.

~~The following are two example URLs~~ Example 2: OData URL broken down into ~~their~~its component parts:

```
~~http://services.odata.org/OData/OData.svc/~~
~~\_______________________________________/~~
~~                 |~~
~~         service root URL~~

~~http://services.odata.org/OData/OData.svc/Categories(1)/Products?$top=2&$orderby=Name~~
~~\_______________________________________/\____________________/ \__________________/~~
~~                 |                                  |                        |~~
~~         service root URL                    resource path           query options~~
http://host:port/path/SampleService.svc/Categories(1)/Products?$top=2&$orderby=Name
\                                      /\                    / \                  /
                  |                                |                       |
          service root URL                  resource path           query options
```

Mandated and suggested content of these three significant URL components used by an OData service are covered in sequence in the three following chapters.

OData follows the URI syntax rules defined in **~~[RFC3986]~~[RFC3986]** and in addition assigns special meaning to several of the sub-delimiters defined by **~~[RFC3986]~~[RFC3986]**, so special care has to be taken regarding parsing and percent-decoding.

~~[RFC3986]~~[RFC3986] defines three steps for URL processing that MUST be performed before percent-decoding:

- Split undecoded URL into components scheme, hier-part, query, and fragment at first "`:`", then first "`?`", and then first "`#`"
- Split undecoded hier-part into authority and path
- Split undecoded path into path segments at "`/`"

After applying these steps defined by RFC3986 the following steps MUST be performed:

- Split undecoded query at "`&`" into query options, and each query option at the first "`=`" into query option name and query option value
- Percent-decode path segments, query option names, and query option values
- Interpret path segments, query option names, and query option values according to OData rules

One of these rules is that single quotes within string literals are represented as two consecutive single quotes.

~~Examples for~~Example 3: valid OData URLs:

```
http://host/service/People('O''Neil')
http://host/service/People(%27O%27%27Neil%27)
http://host/service/People%28%27O%27%27Neil%27%29
http://host/service/Categories('Smartphone%2FTablet')
```

~~Examples for~~Example 4: invalid OData URLs:

```
http://host/service/People('O'Neil')
http://host/service/People('O%27Neil')
http://host/service/Categories('Smartphone/Tablet')
```

The first and second examples are invalid because a single quote in a string literal must be represented as two consecutive single quotes. The third example is invalid because forward slashes are interpreted as path segment separators and `Categories('Smartphone` is not a valid OData path segment, nor is `Tablet')`.

3 Service Root URL

The service root URL identifies the root of an OData service. ~~This URL MUST point to an AtomPub Service Document (as specified in **[RFC5023]**).~~A `GET` request to this URL returns the format-specific service document, see **[OData-JSON]** and **[OData-Atom]**.

~~Per default this~~The service document ~~MUST follow the OData conventions for AtomPub Service Documents. If a different format has been explicitly requested, a corresponding alternate representation of an AtomPub Service Document MUST be delivered. **[OData-JSON]** specifies such an alternate JSON-based representation of a service document.~~

~~Regardless of the format, the service document is required to be returned from the root of an OData service to enable~~enables simple hypermedia-driven clients to enumerate and explore the resources ~~offered~~published by the ~~d~~ODdata service.

4 Resource Path

The rules for resource path construction as defined in this section are optional. OData services SHOULD follow the subsequently described URL path construction rules and are indeed encouraged to do so; as such consistency promotes a rich ecosystem of reusable client components and libraries.

~~**Note:** The query string rules described in the next chapter are required and MUST be followed by any OData service.~~

~~Any aspect of any resource~~Services that do not follow the resource path conventions for entity container children are strongly encouraged to document their resource paths by annotating entity container children with the term `Core.ResourcePath` defined in **[OData-VocCore]**. The annotation value is the URL of the annotated resource and may be relative to `xml:base` (if present), otherwise `Content-Location` (if present), otherwise the request URL.

Resources exposed by an OData service ~~MUST be~~are addressable by ~~a~~ corresponding resource path URL components to enable interaction of the client with that resource aspect.

To illustrate the con~~t~~ce~~x~~pt, some examples for resources might be: ~~C~~customers, a single ~~Customer, Orders~~customer, orders related to a single ~~C~~customer, and so forth. Examples of addressable aspects of these resources as exposed by the data model might be: collections of entities, a single entity, properties, links, operations, and so on.

An OData service MAY respond with `301 Moved Permanently` or `307 Temporary Redirect` from the canonical URL to the actual URL.

4.1 Addressing the Model for a Service

OData services ~~SHOULD~~ expose their ~~Entity Model~~entity model according to ~~**[OData-CSDL]**~~**[OData-CSDL]** at the metadata URL, formed by appending `$metadata` to the ~~Service Root URL.~~service root URL.

~~For example:~~

Example 5: Metadata document URL

`http://`~~`services.odata.org/OData/OData.svc`~~`host/service/$metadata`

OData services MAY expose their entity model as a service, according to **[OData-CSDL]**, by appending a trailing slash (`/`) to the metadata document URL.

Example 6: Metadata service root URL

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

4.2 Addressing the Batch Endpoint for a Service

OData services that support batch requests expose a batch URL formed by appending `$batch` to the ~~Service Root URL.~~service root URL.

~~For example:~~

Example 7: batch URL

`http://`~~`services.odata.org/OData/OData.svc`~~`host/service/$batch`

4.3 Addressing Entities

The basic rules for addressing a collection (of entities), a single entity within a collection, a ~~named entity~~singleton, as well as a property of an entity are covered in the `resourcePath` syntax rule in **[OData-ABNF]**.

Below is a (non-normative) snippet from **[OData-ABNF]**:

```
resourcePath = ~~[ containerQualifier ]~~ entitySetName
[collectionNavigation]
```

```
                 ~~/ [ containerQualifier ] entityName~~
                 / singleton                            [singleNavigation]
                 / actionImportCall
                 / entityColFunctionImportCall          [ collectionNavigation ]
                 / entityFunctionImportCall             [ singleNavigation ]
                 / complexColFunctionImportCall         [ collectionPath ]
                 / complexFunctionImportCall            [ complexPath ]
                 / primitiveColFunctionImportCall       [ collectionPath ]
                 / primitiveFunctionImportCall          [ singlePath ]
                 / crossjoin
                 / '$all'
```

Since OData has a uniform composable URL syntax and associated rules there are many ways to address a collection of entities, including, but not limited to:

- Via an entity set (see rule `entitySetName` in **[OData-ABNF]**)

  ~~For example:~~

```
~~http://services.odata.org/OData/OData.svc/Products~~
```

Example 8:

```
http://host/service/Products
```

- By navigating a collection-valued navigation property (see rule: `entityColNavigationProperty`)
- By invoking a function that returns a collection of entities (see rule: `entityColFunctionCall`)

  ~~For example:~~

Example 9: function with parameters in resource path

```
http://~~services.odata.org/OData/OData.svc~~host/service/ProductsByCategoryId(
categoryId=2)
```

Example 10: function with parameters as query options

```
http://~~services.odata.org/OData/OData.svc~~host/service/ProductsByColor~~?~~(colo
r=@color)?@color='red'
```

- By invoking an action that returns a collection of entities (see rule: `actionCall`)

Likewise there are many ways to address a single entity.

Sometimes a single entity can be accessed directly, for example by:

- Invoking a function that returns a single entity (see rule: `entityFunctionCall`)
- Invoking an action that returns a single entity (see rule: `actionCall`)
- Addressing a ~~named entity~~singleton

~~For example:~~

Example 11:

```
http://~~services.odata.org/OData/OData.svc~~host/service/BestProductEverCreate
d
```

Often however a single entity is accessed by composing more path segments to a `resourcePath` that identifies a collection of entities, for example by:

- Using an entity key to select a single entity (see rules: `collectionNavigation` and `keyPredicate`)

  ~~For example:~~

  ~~*http://services.odata.org/OData/OData.svc/Categories(1)*~~*Example 12:*

```
http://host/service/Categories(1)
```

- Invoking an action bound to a collection of entities that returns a single entity (see rule: `boundOperation`)
- Invoking an function bound to a collection of entities that returns a single entity (see rule: `boundOperation`)

  ~~*For example:*~~*Example 13:*

```
http://~~services.odata.org/OData/OData.svc~~host/service/Products/Model.MostExpensive()
```

These rules are recursive, so it is possible to address a single entity via another single entity, a collection via a single entity and even a collection via a collection; examples include, but are not limited to:

- By following a navigation from a single entity to another related entity (see rule: `entityNavigationProperty`)

  ~~For example:~~

  ~~*http://services.odata.org/OData/OData.svc/Products(1)/Supplier*~~*Example 14:*

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

- By invoking a function bound to a single entity that returns a single entity (see rule: `boundOperation`)

~~*For example:*~~*Example 15:*

```
http://~~services.odata.org/OData/OData.svc~~host/service/Products(1)/Model.MostRecentOrder()
```

- By invoking an action bound to a single entity that returns a single entity (see rule: `boundOperation`)
- By following a navigation from a single entity to a related collection of entities (see rule: `entityColNavigationProperty`)

  ~~For example:~~

~~*http://services.odata.org/OData/OData.svc/Categories(1)/Products*~~*Example 16:*

```
http://host/service/Categories(1)/Products
```

- By invoking a function bound to a single entity that returns a collection of entities (see rule: `boundOperation`)

~~*For example:*~~*Example 17:*

```
http://~~services.odata.org/OData/OData.svc~~host/service/Categories(1)/Model.TopTenProducts()
```

- By invoking an action bound to a single entity that returns a collection of entities (see rule: `boundOperation`)
- By invoking a function bound to a collection of entities that returns a collection of entities (see rule: `boundOperation`)

~~For example:~~Example 18:

```
http://~~services.odata.org/OData/OData.svc~~host/service/Categories(1)/Products/Mo
del.AllOrders()
```

- By invoking an action bound to a collection of entities that returns a collection of entities (see rule: `boundOperation`)

Finally it is possible to compose path segments onto a resource path that identifies a primitive, complex instance, collection of primitives or collection of complex instances and bind an action or function that returns an entity or collections of entities.

4.3.1 Canonical URL

For OData services conformant with the addressing conventions in this section, the canonical form of an absolute URL identifying a non-contained entity is formed by adding a single path segment to the service root URL. The path segment is made up of the name of the entity set associated with the entity followed by the key predicate identifying the entity within the collection. No type-cast segment is added to the canonical URL, even if the entity is an instance of a type derived from the declared entity type of its entity set.

~~For example the URLs~~

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

~~and~~

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

~~both represent the same entity, but the~~ Example 19: Non-canonical URL

```
http://host/service/Categories(1)/Products(1)
```

Example 20: Canonical URL for ~~the entity is the latter.~~previous example:

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

4.3.2 Canonical URL for Contained Entities

For contained entities (i.e. related via a navigation property that specifies `ContainsTarget="true"`, see **[OData-CSDL]**) the canonical URL is the canonical URL of the ~~parent appended~~containing entity followed by:

- A ~~p~~cast~~h~~ segment ~~containing the path of~~if the navigation property is defined on a type derived from the entity type declared for the entity set,
- A path segment for the containment navigation property, and
- If the navigation property returns a collection, ~~the~~a key predicate that uniquely identifies the entity in that collection.

~~4.3.2~~4.3.3 URLs for Related Entities with Referential Constraints

If a navigation property leading to a related entity type has a partner navigation property that specifies a referential constraint, then those key properties of the related entity that take part in the referential constraint MAY be omitted from URLs.

~~Example:~~

Example 21: full key predicate of related entity

```
https://host/service/Orders(1)/Items(OrderID=1,ItemNo=2)
```

~~and~~

Example 22: shortened key predicate of related entity

```
https://host/service/Orders(1)/Items(2)
```

The two above examples are equivalent if the navigation property `Items` from `Order` to `OrderItem` has a partner navigation property from `OrderItem` to `Order` with a referential constraint tying the value of the `OrderID` key property of the `OrderItem` to the value of the `ID` property of the `Order`.

The shorter form that does not specify the constrained key parts redundantly is preferred. If the value of the constrained key is redundantly specified then it MUST match the principal key value.

4.3.4 Resolving an Entity-Id

To resolve an entity-id 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 URL. The entity-id MUST be specified using the system query option `$id`. The entity-id may be expressed as an absolute IRI or relative to the service root URL.

Example 23: request the entity representation for an entity-id

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

The semantics of `$entity` are covered in the **[OData-Protocol]** document.

4.4 Addressing References between Entities

OData services are based on a data model that supports relationships as first class constructs. For example, an OData service could expose a collection of Products entities each of which are related to a Category entity.

References between entities are addressable in OData just like entities themselves are (as described above) by appending a navigation property name followed by `/$ref` to the entity URL.

~~The~~ Example 24: URL ~~given in the following example addresses~~addressing the references between `Categories(1)` and `Products`~~.~~

```
http://~~services.odata.org/OData/OData.svc~~host/service/Categories(1)/Products/$
ref
```

Resource paths addressing a single entity reference can be used in `DELETE` requests to unrelated two entities. Resource paths addressing collection of references can be used in `DELETE` requests if they are followed by the system query option $id identifying one of the entity references in the collection. The entity-id specified by $id may be expressed absolute or relative to the request URL. For details see **[OData-Protocol]**.

Example 25: two ways of unrelating `Categories(1)` and `Products('Bread')`

```
DELETE http://host/service/Categories(1)/Products/$ref?$id=Products('Bread')
DELETE http://host/service/Products('Bread')/Category/$ref
```

4.5 Addressing Operations

4.5.1 Addressing Actions

The semantic rules for addressing and invoking actions are defined in the **[OData-Protocol]** document. The grammar for addressing and invoking actions is defined by the following syntax grammar rules in **[OData-ABNF]**:

- The `actionImportCall` syntax rule defines the grammar in the `resourcePath` for addressing and invoking an action import directly from the service root.
- The `boundActionCall` syntax rule defines the grammar in the `resourcePath` for addressing and invoking an action that is appended to a `resourcePath` that identifies some resources that ~~should~~can be used as the binding parameter value when invoking the action.
- The `boundOperation` syntax rule (which encompasses the `boundActionCall` syntax rule), when used by the `resourcePath` syntax rule, illustrates how a `boundActionCall` can be appended to a `resourcePath`.

4.5.2 Addressing Functions

The semantic rules for addressing and invoking functions are defined in the **[OData-Protocol]** document. The grammar for addressing and invoking functions is defined by a number syntax grammar rules in **[OData-ABNF]**, in particular:

- The `xxxFunctionImportCall` syntax rules define the grammar in the `resourcePath` for addressing and providing parameters for a function import directly from the service root.
- The `boundXxxFunctionCall` syntax rules define the grammar in the `resourcePath` for addressing and providing parameters for a function that is appended to a `resourcePath` that identifies some resources that ~~should~~can be used as the binding parameter value when invoking the function.
- The `boundOperation` syntax rule (which encompasses the `boundXxxFunctionCall` syntax rules), when used by the `resourcePath` syntax rule, illustrates how a `boundXxxFunctionCall` can be appended to a `resourcePath`.
- The `functionExpr`, `boolFunctionExpr`, and `boundFunctionExpr` syntax rules as used by the `filter` and `orderby` syntax rules define the grammar for invoking functions to help filter and order resources identified by the `resourcePath` of the URL.
- The `aliasAndValue` syntax rule defines the grammar for providing function parameter values using Parameter Alias Syntax~~ **[OData-Protocol, 7.4.2.3.2]**.~~, see **[OData-Protocol]**.
- ~~The `parameterNameAndValue` syntax rule defines the grammar for providing function parameter values using Parameter Name Syntax **[OData-Protocol, 7.4.2.3.2]**.~~

4.6 Addressing a Property

To address an entity property clients append a path segment containing the property name to the URL of the entity. If the property has a complex type value, properties of that value can be addressed by further property name composition.

4.7 Addressing a Property Value

To address the raw value of a primitive property, clients append a path segment containing the string `$value` to the property URL.

~~This is not possible for named resource streams, i.e. properties~~Properties of type `Edm.Stream`~~, as these~~ already return the raw value of the media stream ~~without~~and do not support appending the `$value` segment.

4.8 Addressing the Count of ~~an Entity Set or~~ a Collection

To address the raw value of the number of ~~entries~~items in a ~~set or~~ collection, clients append ~~a~~the path segment ~~containing the string $~~/`$count` to the URL identifying the entity set or collection~~ property URL~~.

~~For example~~

~~`http://services.odata.org/OData/OData.svc/Categories(1)/Products/$count`~~

~~and~~

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

~~This can also be used in `$filter` and `$orderby` expressions:~~

~~`http://services.odata.org/OData/OData.svc/Categories?$filter=Products/$count gt 0`~~

~~and~~

~~http://services.odata.org/OData/OData.svc~~Example 26: the number of related entities

```
http://host/service/Categories(1)/Products/$count
```

Example 27: the number of entities in an entity set

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

Example 28: entity count in a `$filter` expression. Note that the spaces around `gt` are for readability of the example only; in real URLs they must be percent-encoded as `%20`.

```
http://host/service/Categories?$filter=Products/$count gt 0
```

Example 29: entity count in an `$orderby` expression

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

4.9 Addressing Derived Types

Any resource path or path expression identifying a collection of entities or complex type instances ~~may~~can be appended with a path segment containing the qualified name of a type derived from the declared type of the collection. The result will be restricted to instances of the derived type and may be empty.

Any resource path or path expression identifying a single entity or complex type instance ~~may~~can be appended with a path segment containing the qualified name of a type derived from the declared type of the identified resource. If used in a resource path and the identified resource is not an instance of the derived type, the request will result in a `404 Not Found` ~~error~~response. If used in a path expression that is part of a ~~b~~Boolean expression, the ~~boolean expression~~type cast will evaluate to ~~`false`~~`null`.

~~For example~~

Example 30: entity set restricted to `VipCustomer` instances

```
http://host/service/Customers/Model.VipCustomer
```

~~will restrict the result to `VipCustomer` instances.~~

Example 31: entity restricted to a `VipCustomer` instance, resulting in `404 Not Found` if the customer with key `1` is not a `VipCustomer`

```
http://host/service/Customers/Model.VipCustomer(1)
http://host/service/Customers(1)/Model.VipCustomer
```

~~will result in `404 Not Found` if the customer with key `1` is not a `VipCustomer`.~~

~~`http://host/service/Customers(1)/Address/Model.DetailedAddress/Location`~~

~~will~~Example 32: cast the complex property `Address` to its derived type `DetailedAddress`, then get a property of the derived type~~.~~

```
http://host/service/Customers~~?$filter=Model.VipCustomer/PercentageOfVipPromoti
onProductsOrdered gt 80~~(1)/Address/Model.DetailedAddress/Location
```

Example 33: filter expression with type cast; will evaluate to ~~`false`~~`null` for all non-`VipCustomer` instances and thus return only instances of `VipCustomer`~~.~~

```
http://host/service/~~Orders?$~~Customers?
        $filter=Model.VipCustomer/PercentageOfVipPromotionProductsOrdered gt 80
Example 34: expand~~=Customer/Model.VipCustomer~~
```

~~will inline~~ the single related `Customer` only if it is an instance of `Model.VipCustomer`. For to-many relationships only `Model.VipCustomer` instances would be inlined~~.~~,

```
http://host/service/Orders?$expand=Customer/Model.VipCustomer
```

4.10 Addressing the Media Stream of a Media Entity

To address the media stream represented by a media entity, clients append a path segment containing the string `$value` to the media entity URL. Services ~~MAY~~may redirect from this canonical URL to the source URL of the media stream.

~~For example a `GET`~~Example 35: request ~~to the following URL will return, or redirect to a URL that returns,~~ the media stream for the picture with the key value ~~"~~`Sunset4321299432`~~".~~:

```
http://host/service/Pictures('Sunset4321299432')/$value
```

4.11 Addressing the Cross Join of Entity Sets

In addition to querying related entities through navigation properties defined in the entity model of a service, the cross join operator allows querying across unrelated entity sets.

The cross join is addressed by appending the path segment `$crossjoin` followed by the parenthesized comma-separated list of joined entity sets. It returns the Cartesian product of all the specified entity sets, represented as a collection of instances of a virtual complex type. Each instance consists of one non-nullable, single-valued navigation property per joined entity set. Each such navigation property is named identical to the corresponding entity set, with a target type equal to the declared entity type of the corresponding entity set.

The `$filter` and `$orderby` query options can be specified using properties of the entities in the selected entity sets, prepended with the entity set as the navigation property name.

The `$expand` query option can be specified using the names of the selected entity sets as navigation property names. If a selected entity set is not expanded, it MUST be represented using the read URL of the related entity as a navigation link in the complex type instance.

The `$count`, `$skip`, and `$top` query options can also be used with no special semantics.

Example 36: if `Sales` had a structural property `ProductID` instead of a navigation property `Product`, a “cross join” between `Sales` and `Products` could be addressed

```
http://host/service/$crossjoin(Products,Sales)?
                                $filter=Products/ID eq Sales/ProductID
```

4.12 Addressing All Entities in a Service

The symbolic resource `$all` identifies the collection of all entities in a service, i.e. the union of all entity sets plus all singletons.

This symbolic resource is of type `Collection(Edm.EntityType)` and allows the `$search` system query option plus all other query options applicable to collections of entities.

Example 37: all entities in a service

```
http://host/service/$all
```

5 Query Options

The ~~Query Options section~~query options part of an OData URL specifies three types of information: ~~System Query Options, Custom Query Options, and Function Parameters.~~system query options, custom query options, and parameter aliases. All OData services MUST follow the query string parsing and construction rules defined in this section and its subsections.

5.1 System Query Options

System ~~Query Options~~query options are query string parameters ~~a client may specify to~~ that control the amount and order of the data ~~that an OData service returns~~ returned for the resource identified by the URL. The names of all ~~System Query Options~~system query options are prefixed with a ~~“$”~~dollar ($) character.

For GET requests the following rules apply:

- Resource paths identifying a single entity, a complex type instance, a collection of entities, or a collection of ~~entities~~complex type instances allow ~~$expand~~$expand and ~~$select.~~$select.
- Resource paths identifying a collection allow $filter, $count, $orderby, $skip, and $top.
- Resource paths identifying a collection of entities allow ~~$filter, $count, $orderby, $skip, and $top.~~ $search.
- ~~All resource~~Resource paths not ending in /$~~value, /$~~count~~,~~ or /$~~metadata~~batch allow ~~$format~~$format.

For POST requests to an action URL the return type of the action determines the applicable system query options, following the same rules as GET requests.

POST requests to entity sets as well as all PUT and DELETE requests do not allow system query options.

An OData service may support some or all of the ~~System Query Options~~system query options defined. If a data service does not support a ~~System Query Option~~system query option, it ~~must~~MUST reject any request that contains the unsupported option.

The semantics of all ~~System Query Options~~system query options are defined in the **[OData-Protocol]** document.

The grammar and syntax rules for ~~System Query Options~~system query options are defined in **[OData-ABNF]**.

5.1.1 ~~Filter~~ System Query Option $filter

The $filter system query option allows clients to filter ~~the set~~a collection of resources that are addressed by a request URL. The expression specified with $filter ~~specifies conditions that MUST be met by a resource~~is evaluated for ~~it to be returned~~ each resource in the ~~set of matching resources~~collection, and only items where the expression evaluates to true are included in the response. Resources for which the expression evaluates to false or to null, or which reference properties that are unavailable due to permissions, are omitted from the response.

The **[OData-ABNF]** filter syntax rule defines the formal grammar of the $filter query option.

5.1.1.1 Logical Operators

OData defines a set of logical operators that evaluate to true or false (i.e. a boolCommonExpr as defined in **[OData-ABNF]**). Logical operators are typically used ~~in the Filter System Query Option to filter the set of resources. However services MAY allow for the use of Logical Operators with the OrderBy System Query Option~~to filter a collection of resources.

Operands of collection, entity, and complex types are not supported in logical operators.

The syntax rules for the logical operators are defined in **[OData-ABNF]**.

The six comparison operators can be used with all primitive values except `Edm.Binary`, `Edm.Stream`, and the `Geo` types.

5.1.1.1.1 Equals ~~Operator~~

The ~~Equals~~`eq` operator ~~(or “`eq`”) evaluates” to~~returns true if the left operand is equal to the right operand, otherwise ~~if evaluates to~~ it returns false.

The `null` value is equal to itself, and only to itself.

5.1.1.1.2 Not Equals ~~Operator~~

The ~~Not Equals~~`ne` operator ~~(or “`ne`”) evaluates to~~returns true if the left operand is not equal to the right operand, otherwise ~~if evaluates to~~it returns false.

The `null` value is not equal to any value but itself.

5.1.1.1.3 Greater Than ~~Operator~~

The ~~Greater Than~~`gt` operator ~~(or “`gt`”) evaluates to~~returns true if the left operand is greater than the right operand, otherwise ~~if evaluates to~~it returns false.

If any operand is `null`, the operator returns false.

For Boolean values true is greater than false.

5.1.1.1.4 Greater Than or Equal ~~Operator~~

~~The Greater Than or Equal operator (or “~~The `ge`~~”) evaluates to~~ operator returns true if the left operand is greater than or equal to the right operand, otherwise ~~if evaluates to~~it returns false.

If only one operand is `null`, the operator returns false. If both operands are `null`, it returns true because `null` is equal to itself.

5.1.1.1.5 Less Than ~~Operator~~

The ~~Less Than operator (or “~~`lt`~~”) evaluates to~~ operator returns true if the left operand is less than the right operand, otherwise ~~if evaluates to~~it returns false.

If any operand is `null`, the operator returns false.

5.1.1.1.6 Less Than or Equal ~~Operator~~

The ~~Less Than operator (or “~~`le`~~”) evaluates to~~ operator returns true if the left operand is less than or equal to the right operand, otherwise ~~if evaluates to~~it returns false.

~~Logical~~ If only one operand is `null`, the operator returns false. If both operands are `null`, it returns true because `null` is equal to itself.

5.1.1.1.7 And ~~Operator~~

The ~~Logical And~~`and` operator ~~(or “`and`”) evaluates to~~returns true if both the left and right operands ~~both~~ evaluate to true, otherwise ~~if evaluates to~~it returns false.

~~5.1.1.1.8 Logical Or Operator~~

~~The Logical Or operator (or “`or`”) evaluates to~~The `null` value is treated as unknown, so if one operand evaluates to `null` and the other operand to false, the `and` operator returns false. All other combinations with `null` return `null`.

5.1.1.1.8 Or

The or operator returns false if both the left and right operands both evaluate to false, otherwise ~~if evaluates to~~it returns true.

~~5.1.1.1.9 Logical Negation Operator~~

The ~~Logical Negation Operator (or “not”) evaluates to true~~null value is treated as unknown, so if ~~the~~one operand evaluates to null and the other operand to true, the or operator returns true. All other combinations with null return null.

5.1.1.1.9 Not

The not operator returns true if the operand returns false, otherwise it ~~evalutes to~~returns false.

The null value is treated as unknown, so not null returns null.

5.1.1.1.10 has

The has operator returns true if the right hand operand is an enumeration value whose flag(s) are set on the left operand.

~~5.1.1.1.10~~5.1.1.1.11 Logical Operator Examples

The following examples illustrate the use and semantics of each of the logical operators. ~~They contain unencoded spaces to increase readability. In real life the spaces would need to be encoded as %20, which most browsers will do anyway if a space is entered in the address bar.~~

~~http://services.odata.org/OData/OData.svc/Products?$filter=Name eq 'Milk'~~

~~(Requests~~Example 38: all products with a Name equal to 'Milk'~~)~~

~~http://services.odata.org/OData/OData.svc/Products?$filter=Name ne 'Milk'~~

~~(Requests~~http://host/service/Products?$filter=Name eq 'Milk'

Example 39: all products with a Name not equal to 'Milk'~~).~~

~~http://services.odata.org/OData/OData.svc/Products?$filter=Name gt 'Milk'~~

~~(Requests~~http://host/service/Products?$filter=Name ne 'Milk'

Example 40: all products with a Name greater than 'Milk'~~).~~:

~~http://services.odata.org/OData/OData.svc/Products?$filter=Name ge 'Milk'~~

~~(Requests~~http://host/service/Products?$filter=Name gt 'Milk'

Example 41: all products with a Name greater than or equal to 'Milk'~~).~~:

~~http://services.odata.org/OData/OData.svc/Products?$filter=Name lt 'Milk'~~

~~(Requests~~http://host/service/Products?$filter=Name ge 'Milk'

Example 42: all products with a Name less than 'Milk'~~).~~:

~~http://services.odata.org/OData/OData.svc/Products?$filter=Name le 'Milk'~~

~~(Requests~~http://host/service/Products?$filter=Name lt 'Milk'

Example 43: all products with a Name less than or equal to 'Milk'~~).~~:

~~http://services.odata.org/OData/OData.svc/Products?$filter=Name eq 'Milk' and Price lt '2.55'~~

~~(Requests~~`http://host/service/Products?$filter=Name le 'Milk'`

Example 44: all products with the Name 'Milk' that also have a Price less than 2.55~~).~~:

~~`http://services.odata.org/OData/OData.svc/Products?$filter=Name eq 'Milk' or Price lt 2.55`~~

~~(Requests~~`http://host/service/Products?$filter=Name eq 'Milk' and Price lt '2.55'`

Example 45: all products that either have the Name 'Milk' or have a Price less than 2.55~~).~~:

~~`http://services.odata.org/OData/OData.svc/Products?$filter=not endswith(Name,'ilk')`~~

~~(Requests~~`http://host/service/Products?$filter=Name eq 'Milk' or Price lt 2.55`

Example 46: all products that do not have a Name that ends with 'ilk'~~).~~ :

```
http://host/service/Products?$filter=not endswith(Name,'ilk')
```

5.1.1.2 Arithmetic Operators

OData defines a set of arithmetic operators that require operands that evaluate to numeric types. Arithmetic ~~O~~operators are typically used ~~in the Filter System Query Option~~ to filter ~~the set~~a collection of resources. However services MAY allow ~~for the use of Arithmetic Operators~~using arithmetic operators with the ~~OrderBy System Query Option.~~`$orderby` system query option.

If an operand of an arithmetic operator is null, the result is null.

The syntax rules for the ~~Arithmetic Operators~~arithmetic operators are defined in **[OData-ABNF]**.

5.1.1.2.1 Addition ~~Operator~~

The ~~Addition Operator (or "~~`add`~~")~~ operator adds the left and right numeric operands ~~together~~.

The `add` operator is also valid for the following time-related operands:

- `DateTimeOffset add Duration` results in a `DateTimeOffset`
- `Duration add Duration` results in a `Duration`
- `Date add Duration` results in a `DateTimeOffset`

5.1.1.2.2 Subtraction ~~Operator~~

The ~~Subtraction Operator (or "~~`sub`~~")~~ operator subtracts the right numeric operand from the left numeric operand.

The `sub` operator is also valid for the following time-related operands:

- `DateTimeOffset sub Duration` results in a `DateTimeOffset`
- `Duration sub Duration` results in a `Duration`
- `DateTimeOffset sub DateTimeOffset` results in a `Duration`
- `Date sub Duration` results in a `DateTimeOffset`
- `Date sub Date results` in a `Duration`

5.1.1.2.3 Negation ~~Operator~~

The ~~Negation Operator (or "–")~~negation operator, represented by a minus (–) sign, changes the sign of its numeric or `Duration` operand.

5.1.1.2.4 Multiplication ~~Operator~~

The ~~Multiplication Operator (or "~~`mul`~~")~~ operator multiplies the left and right numeric operands ~~together~~.

5.1.1.2.5 Division ~~Operator~~

The ~~Division Operator (or “div”)~~ operator divides the left numeric operand by the right numeric operand. If the right operand is zero and the left operand is neither of type Edm.Single nor Edm.Double, the request fails. If the left operand is of type Edm.Single or Edm.Double, then positive div zero returns INF, negative div zero returns –INF, and zero div zero returns = NaN.

5.1.1.2.6 Modulo ~~Operator~~

The ~~Modulo Operator (or “mod”) evaluates to~~ operator returns the remainder when the left integral operand is divided by the right integral operand. If the right operand is negative, the sign of the result is the same as the sign of the left operand. If the right operand is zero, the request fails.

5.1.1.2.7 Arithmetic Operator Examples

The following examples illustrate the use and semantics of each of the Arithmetic operators~~:~~.

~~http://services.odata.org/OData/OData.svc/Products?$filter=Price add 2.45 eq 5.00~~

~~(Requests~~Example 47: all products with a Price of 2.55~~).~~:

~~http://services.odata.org/OData/OData.svc/Products?$filter=Price sub 0.55 eq 2.00~~

~~(Requests~~`http://host/service/Products?$filter=Price add 2.45 eq 5.00`

Example 48: all products with a Price of 2.55:

```
http://host/service/Products?$filter=Price sub 0.55 eq 2.00
```

Example 49: all products with a Price of 2.55~~).~~:

~~http://services.odata.org/OData/OData.svc/Products?$filter=Price mul 2.0 eq 5.10~~

~~(Requests~~`http://host/service/Products?$filter=Price mul 2.0 eq 5.10`

Example 50: all products with a Price of 2.55~~).~~:

~~http://services.odata.org/OData/OData.svc/Products?$filter=Price div 2.55 eq 1~~

~~(Requests all products with a Price of 2.55).~~

~~http://services.odata.org/OData/OData.svc/Products?$filter=Rating mod 5 eq 0~~

~~(Requests~~`http://host/service/Products?$filter=Price div 2.55 eq 1`

Example 51: all products with a Rating exactly divisible by 5~~).~~:

```
http://host/service/Products?$filter=Rating mod 5 eq 0
```

5.1.1.3 Grouping ~~Operator~~

The Grouping ~~O~~operator (open and close parenthesis "( )") controls the evaluation order of an expression. The Grouping ~~Operator evaluates to~~operator returns the expression grouped inside the parenthesis. ~~For example:~~

~~http://services.odata.org/OData/OData.svc/Products?$filter=(4 add 5) mod (4 sub 1) eq 0~~

~~Requests~~Example 52: all products~~,~~ because 9 mod 3 is 0~~.~~

```
http://host/service/Products?$filter=(4 add 5) mod (4 sub 1) eq 0
```

5.1.1.4 Canonical Functions

In addition to operators, a set of functions is also defined for use with the $filter or ~~orderby~~$orderby system query options. The following ~~table lists~~sections describe the available functions. Note: ISNULL or COALESCE operators are not defined. Instead, ~~there is~~ OData defines a ~~null~~null literal that can be used in comparisons.

If a parameter of a canonical function is null, the function returns null.

The syntax rules for all ~~canonical~~ functions are defined in **[OData-ABNF]**.

~~5.1.1.4.1 substringof~~

5.1.1.4.1 contains

The ~~substringof canonical~~contains function has the following signature:

Edm.Boolean ~~substringof~~contains(Edm.String,Edm.String)

~~If implemented the substringof canonical~~The contains function ~~MUST return~~returns true if~~, and only if, the first parameter string value is a substring of the~~ the second parameter string value~~. The substringOfMethodCallExpr~~ is a substring of the first parameter string value, otherwise it returns false. The containsMethodCallExpr syntax rule defines how the ~~substringof~~contains function is invoked.

~~For example:~~

~~http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=substringof('Alfreds',CompanyName)~~

~~Returns~~Example 53: all customers with a CompanyName that contains 'Alfreds'~~.~~

```
http://host/service/Customers?$filter=contains(CompanyName,'Alfreds')
```

5.1.1.4.2 endswith

The endswith ~~canonical~~ function has the following signature:

Edm.Boolean endswith(Edm.String,Edm.String)

~~If implemented the~~ The endswith ~~canonical~~ function ~~MUST~~ returns true if~~, and only if,~~ the first parameter string value ends with the second parameter string value~~.~~, otherwise it returns false. The endsWithMethodCallExpr syntax rule defines how the endswith function is invoked.

~~For example:~~

~~http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=endswith(CompanyName,'Futterkiste')~~

~~Returns~~Example 54: all customers with a CompanyName that ends with 'Futterkiste'~~.~~

```
http://host/service/Customers?$filter=endswith(CompanyName,'Futterkiste')
```

5.1.1.4.3 startswith

The startswith ~~canonical~~ function has the following signature:

Edm.Boolean startswith(Edm.String,Edm.String)

~~If implemented the~~ The `startswith` ~~canonical~~ function ~~MUST return~~returns `true` if~~, and only if,~~ the first parameter string value starts with the second parameter string value~~.~~, otherwise it returns `false`. The `startsWithMethodCallExpr` syntax rule defines how the `startswith` function is invoked.

~~For example:~~

~~http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=startswith(CompanyName,'Alfr')~~

~~Returns~~Example 55: all customers with a `CompanyName` that starts with `'Alfr'`

```
http://host/service/Customers?$filter=startswith(CompanyName,'Alfr')
```

5.1.1.4.4 `length`

The `length` ~~canonical~~ function has the following signature:

```
Edm.Int32 length(Edm.String)
```

~~If implemented the~~The `length` ~~canonical~~ function ~~MUST return~~returns the number of characters in the parameter value. The `lengthMethodCallExpr` syntax rule defines how the `length` function is invoked.

~~For example:~~

~~http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=length(CompanyName) eq 19~~

~~Returns~~Example 56: all customers with a `CompanyName` that is 19 characters long~~.~~

```
http://host/service/Customers?$filter=length(CompanyName) eq 19
```

5.1.1.4.5 `indexof`

The ~~length canonical~~`indexof` function has the following signature:

```
Edm.Int32 indexof(Edm.String,Edm.String)
```

~~If implemented the~~The `indexof` ~~canonical~~ function ~~MUST return~~returns the zero-based character position of the first occurrence of the second parameter value in the first parameter value. The `indexOfMethodCallExpr` syntax rule defines how the `indexof` function is invoked.

~~For example:~~

~~http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=indexof(CompanyName,'lfreds') eq 1~~

~~Returns~~Example 57: all customers with a `CompanyName` containing `'lfreds'` starting at the second character~~.~~

```
http://host/service/Customers?$filter=indexof(CompanyName,'lfreds') eq 1
```

5.1.1.4.6 `substring`

The `substring` ~~canonical~~ function has consists of two overloads, with the following signatures:

```
Edm.String substring(Edm.String,Edm.Int32)
Edm.String substring(Edm.String,Edm.Int32,Edm.Int32)
```

~~If implemented the~~The two argument `substring` ~~canonical~~ function ~~MUST return~~returns a substring of the first parameter string value, starting at the Nth character and finishing at the last character (where N is the second parameter integer value). ~~If implemented, the~~The three argument `substring` ~~canonical~~ function ~~MUST return~~returns a substring of the first parameter string value identified by selecting M characters starting at the Nth character (where N is the second parameter integer value and M is the third parameter integer value).

The `substringMethodCallExpr` syntax rule defines how the `substring` ~~canonical~~ functions are invoked.

~~For example:~~

~~`http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=substring(CompanyName, 1) eq 'lfreds Futterkiste'`~~

~~Returns~~Example 58: all customers with a `CompanyName` of `'lfreds Futterkiste'` once the first character has been removed ~~.~~

~~`http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=substring(CompanyName,1,2) eq 'lf'`~~

```
~~Returns~~http://host/service/Customers?
                  $filter=substring(CompanyName, 1) eq 'lfreds Futterkiste'
```

Example 59: all customers with a `CompanyName` that has `'lf'` as the second and third characters~~ respectively.~~

```
http://host/service/Customers?$filter=substring(CompanyName,1,2) eq 'lf'
```

5.1.1.4.7 `tolower`

The `tolower` ~~canonical~~ function has the following signature:

```
Edm.String tolower(Edm.String)
```

~~If implemented the~~The `tolower` ~~canonical~~ function ~~MUST return~~returns the input parameter string value with all uppercase characters converted to lowercase according to Unicode rules. The `toLowerMethodCallExpr` syntax rule defines how the `tolower` function is invoked.

~~For example:~~

~~`http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=tolower(CompanyName) eq 'alfreds futterkiste'`~~

~~Returns~~Example 60: all customers with a `CompanyName` that equals `'alfreds futterkiste'` once any uppercase characters have been converted to lowercase~~.~~

```
http://host/service/Customers?
                     $filter=tolower(CompanyName) eq 'alfreds futterkiste'
```

5.1.1.4.8 `toupper`

The `toupper` ~~canonical~~ function has the following signature:

```
Edm.String toupper(Edm.String)
```

~~If implemented the~~The `toupper` ~~canonical~~ function ~~MUST return~~returns the input parameter string value with all lowercase characters converted to uppercase according to Unicode rules. The `toUpperMethodCallExpr` syntax rule defines how the `to~~low~~upper` function is invoked.

~~For example:~~

~~`http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=toupper(CompanyName) eq 'ALFREDS FUTTERKISTE'`~~

~~Returns~~Example 61: all customers with a `CompanyName` that equals `'ALFREDS FUTTERKISTE'` once any lowercase characters have been converted to uppercase~~.~~

```
http://host/service/Customers?
                         $filter=toupper(CompanyName) eq 'ALFREDS FUTTERKISTE'
```

5.1.1.4.9 trim

The `trim` ~~canonical~~ function has the following signature:

```
Edm.String trim(Edm.String)
```

~~If implemented the~~The `trim` ~~canonical~~ function ~~MUST return~~returns the input parameter string value with all leading and trailing whitespace characters, according to Unicode rules, removed. The `trimMethodCallExpr` syntax rule defines how the `trim` function is invoked.

~~For example:~~

~~`http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=length(trim(CompanyName)) eq length(CompanyName)`~~

~~Returns~~Example 62: all customers with a `CompanyName` without leading or trailing whitespace characters~~.~~

```
http://host/service/Customers?$filter=trim(CompanyName) eq CompanyName
```

5.1.1.4.10 concat

The `concat` ~~canonical~~ function has the following signature:

```
Edm.String concat(Edm.String,Edm.String)
```

~~If implemented the~~ The `concat` ~~canonical~~ function ~~MUST return~~returns a string that appends the second input parameter string value~~s~~ to the first. The `concatMethodCallExpr` syntax rule defines how the `concat` function is invoked.

~~For example:~~

~~`http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=concat(concat(City,', '), Country) eq 'Berlin, Germany'`~~

~~Returns~~Example 63: all customers from ~~the `City` of~~ Berlin~~ and the `Country` called~~, Germany~~.~~

```
http://host/service/Customers?
               $filter=concat(concat(City,', '), Country) eq 'Berlin, Germany'
```

5.1.1.4.11 year

The `year` ~~canonical~~ function has the following signatures:

```
Edm.Int32 year(Edm.Date)
Edm.Int32 year(Edm.DateTimeOffset)
```

~~If implemented the~~The `year` ~~canonical~~ function ~~MUST return~~returns the year component of the `Date` or `DateTimeOffset` parameter value, evaluated in the time zone of the `DateTimeOffset` parameter value. The `yearMethodCallExpr` syntax rule defines how the `year` function is invoked.

~~The `year` function MUST be evaluated in the time zone of the `DateTimeOffset` parameter value.~~

~~For example:~~

~~`http://services.odata.org/Northwind/Northwind.svc/Employees?$filter=year(BirthDate) eq 1971`~~

~~Returns~~Services that are unable to preserve the offset of `Edm.DateTimeOffset` values and instead normalize the values to some common time zone (i.e. UTC) MUST fail evaluation of the `year` function for literal `Edm.DateTimeOffset` values that are not stated in the time zone of the normalized values.

Example 64: all employees ~~who were~~ born in 1971~~.~~

```
http://host/service/Employees?$filter=year(BirthDate) eq 1971
```

5.1.1.4.12 month

The month~~ canonical~~ function has the following signatures:

```
Edm.Int32 month(Edm.Date)
Edm.Int32 month(Edm.DateTimeOffset)
```

~~If implemented the~~The month ~~canonical ~~function ~~MUST return~~returns the month component of the Date or DateTimeOffset parameter value, evaluated in the time zone of the DateTimeOffset parameter value. The monthMethodCallExpr syntax rule defines how the month function is invoked.

~~The month function MUST be evaluated in the time zone of the DateTimeOffset parameter value.~~

~~For example:~~

~~http://services.odata.org/Northwind/Northwind.svc/Employees?$filter=month(BirthDate) eq 5~~

~~Returns~~Services that are unable to preserve the offset of Edm.DateTimeOffset values and instead normalize the values to some common time zone (i.e. UTC) MUST fail evaluation of the month function for literal Edm.DateTimeOffset values that are not stated in the time zone of the normalized values.

Example 65: all employees ~~who were ~~born in May~~.~~

```
http://host/service/Employees?$filter=month(BirthDate) eq 5
```

5.1.1.4.13 day

The day ~~canonical ~~function has the following signatures:

```
Edm.Int32 day(Edm.Date)
Edm.Int32 day(Edm.DateTimeOffset)
```

~~If implemented the~~The day ~~canonical ~~function ~~MUST return~~returns the day component Date or DateTimeOffset parameter value, evaluated in the time zone of the DateTimeOffset parameter value. The dayMethodCallExpr syntax rule defines how the day function is invoked.

~~The day function MUST be evaluated in the time zone of the DateTimeOffset parameter value.~~

~~For example:~~

~~http://services.odata.org/Northwind/Northwind.svc/Employees?$filter=day(BirthDate) eq 8~~

~~Returns~~Services that are unable to preserve the offset of Edm.DateTimeOffset values and instead normalize the values to some common time zone (i.e. UTC) MUST fail evaluation of the day function for literal Edm.DateTimeOffset values that are not stated in the time zone of the normalized values.

Example 66: all employees ~~who were ~~born on the 8th day of a month~~.~~

```
http://host/service/Employees?$filter=day(BirthDate) eq 8
```

5.1.1.4.14 hour

The hour ~~canonical ~~function has the following signatures:

```
Edm.Int32 hour(Edm.DateTimeOffset)
Edm.Int32 hour(Edm.TimeOfDay)
```

~~If implemented the~~The hour ~~canonical ~~function ~~MUST return~~returns the hour component of the DateTimeOffset or TimeOfDay parameter value~~.~~, evaluated in the time zone of the DateTimeOffset parameter value. The hourMethodCallExpr syntax rule defines how the hour function is invoked.

~~The hour function MUST be evaluated in the time zone of the DateTimeOffset parameter value.~~

~~For example:~~

~~http://services.odata.org/Northwind/Northwind.svc/Employees?$filter=hour(BirthDate) eq 4~~

~~Returns~~Services that are unable to preserve the offset of Edm.DateTimeOffset values and instead normalize the values to some common time zone (i.e. UTC) MUST fail evaluation of the hour function for literal Edm.DateTimeOffset values that are not stated in the time zone of the normalized values.

Example 67: all employees ~~who were~~ born in the 4th hour of a day~~.~~

```
http://host/service/Employees?$filter=hour(BirthDate) eq 4
```

5.1.1.4.15 minute

The minute ~~canonical~~ function has the following signatures:

```
Edm.Int32 minute(Edm.DateTimeOffset)
Edm.Int32 minute(Edm.TimeOfDay)
```

~~If implemented the~~The minute ~~canonical~~ function ~~MUST return~~returns the minute component of the DateTimeOffset or TimeOfDay parameter value~~.~~, evaluated in the time zone of the DateTimeOffset parameter value. The minuteMethodCallExpr syntax rule defines how the minute function is invoked.

~~The minute function MUST be evaluated in the time zone of the DateTimeOffset parameter value.~~

~~For example:~~

~~http://services.odata.org/Northwind/Northwind.svc/Employees?$filter=minute(BirthDate) eq 40~~

~~Returns~~Example 68: all employees ~~who were~~ born in the 40th minute of any hour on any day~~.~~

```
http://host/service/Employees?$filter=minute(BirthDate) eq 40
```

5.1.1.4.16 second

The second ~~canonical~~ function has the following signatures:

```
Edm.Int32 second(Edm.DateTimeOffset)
Edm.Int32 second(Edm.TimeOfDay)
```

~~If implemented the~~The second ~~canonical~~ function ~~MUST return~~returns the second component (without the fractional part) of the DateTimeOffset or TimeOfDay parameter value. The secondMethodCallExpr syntax rule defines how the second function is invoked.

~~For example:~~

~~http://services.odata.org/Northwind/Northwind.svc/Employees?$filter=second(BirthDate) eq 40~~

~~Returns~~Example 69: all employees ~~who were~~ born in the 40th second of any minute of any hour on any day~~.~~

```
http://host/service/Employees?$filter=second(BirthDate) eq 40
```

5.1.1.4.17 fractionalseconds

The fractionalseconds ~~canonical~~ function has the following signatures:

```
Edm.Decimal fractionalseconds(Edm.DateTimeOffset)
Edm.Decimal fractionalseconds(Edm.TimeOfDay)
```

~~If implemented the~~The fractionalseconds ~~canonical~~ function ~~MUST return~~returns the fractional seconds component of the DateTimeOffset or TimeOfDay parameter value as a non-negative decimal value ~~smaller~~less than 1. The fractionalsecondsMethodCallExpr syntax rule defines how the fractionalseconds function is invoked.

~~For example:~~

~~http://services.odata.org/Northwind/Northwind.svc/Employees?$filter=fractionalseconds(BirthDate) lt 0.1~~

~~Returns~~Example 70: all employees ~~who were~~ born less than 100 milliseconds after a full second of any minute of any hour on any day~~.~~

~~5.1.1.4.18 totalseconds~~

```
http://host/service/Employees?$filter=fractionalseconds(BirthDate) lt 0.1
```

5.1.1.4.18 date

The ~~totalseconds canonical~~date function has the following signature:

~~Edm.Decimal totalseconds(Edm.Duration)~~

~~If implemented the totalseconds canonical function MUST return the duration of the value in total seconds, including fractional seconds.~~

~~5.1.1.4.19 date~~

~~The date canonical function has the following signature:~~

```
Edm.Date date(Edm.DateTimeOffset)
```

~~If implemented the~~The date ~~canonical~~ function ~~MUST return~~returns the date part of the DateTimeOffset parameter value~~.~~

~~The date function MUST be~~, evaluated in the time zone of the DateTimeOffset parameter value.

~~5.1.1.4.20~~5.1.1.4.19 time

The time ~~canonical~~ function has the following signature:

```
Edm.TimeOfDay time(Edm.DateTimeOffset)
```

~~If implemented the~~The time ~~canonical~~ function ~~MUST return~~returns the time part of the DateTimeOffset parameter value~~.~~

~~The time function MUST be~~, evaluated in the time zone of the DateTimeOffset parameter value.

Services that are unable to preserve the offset of Edm.DateTimeOffset values and instead normalize the values to some common time zone (i.e. UTC) MUST fail evaluation of the time function for literal Edm.DateTimeOffset values that are not stated in the time zone of the normalized values.

~~5.1.1.4.21~~5.1.1.4.20 totaloffsetminutes

The totaloffsetminutes ~~canonical~~ function has the following signature:

```
Edm.Int32 totaloffsetminutes(Edm.DateTimeOffset)
```

~~If implemented the~~The totaloffsetminutes ~~canonical~~ function ~~MUST return~~returns the signed number of minutes in the time zone offset part of the DateTimeOffset parameter value~~.~~

~~The totaloffsetminutes function MUST be~~, evaluated in the time zone of the DateTimeOffset parameter value.

~~5.1.1.4.22~~5.1.1.4.21 now

The now ~~canonical~~ function has the following signature:

```
Edm.DateTimeOffset now()
```

~~If implemented the~~The now ~~canonical~~ function ~~MUST return~~returns the current point in time (date and time with time zone) as a DateTimeOffset value.

Services are free to choose the time zone for the current point, e.g. UTC. Services that are unable to preserve the offset of Edm.DateTimeOffset values and instead normalize the values to some common time zone SHOULD return a value in the normalized time zone (i.e., UTC).

~~5.1.1.4.23~~5.1.1.4.22 maxdatetime

The maxdatetime ~~canonical~~ function has the following signature:

```
Edm.DateTimeOffset maxdatetime()
```

~~If implemented the~~The maxdatetime ~~canonical~~ function ~~MUST return~~returns the latest possible point in time as a DateTimeOffset value.

~~5.1.1.4.24~~5.1.1.4.23 mindatetime

The mindatetime ~~canonical~~ function has the following signature:

```
Edm.DateTimeOffset mindatetime()
```

~~If implemented the~~ The mindatetime ~~canonical~~ function ~~MUST return~~returns the earliest possible point in time as a DateTimeOffset value.

~~5.1.1.4.25 round~~

5.1.1.4.24 totalseconds

The ~~round canonical~~totalseconds function has the following signature:

```
Edm.Decimal totalseconds(Edm.Duration)
```

The totalseconds function returns the duration of the value in total seconds, including fractional seconds.

5.1.1.4.25 round

The round function has the following signatures

```
Edm.Double round(Edm.Double)
Edm.Decimal round(Edm.Decimal)
```

~~If implemented the~~ The round ~~canonical~~ function ~~MUST round~~rounds the input numeric parameter ~~value~~ to the nearest numeric value with no decimal component. The mid-point between two integers is rounded away from zero, i.e. 0.5 is rounded to 1 and -0.5 is rounded to -1. The roundMethodCallExpr syntax rule defines how the round function is invoked.

~~For example:~~

~~http://services.odata.org/Northwind/Northwind.svc/Orders?$filter=round(Freight) eq 32~~

~~Returns~~Example 71: all orders ~~that have a~~with freight costs that round~~s~~ to 32

```
http://host/service/Orders?$filter=round(Freight) eq 32~~.~~
```

5.1.1.4.26 floor

The `floor` ~~canonical~~ function has the following signatures

```
Edm.Double floor(Edm.Double)
Edm.Decimal floor(Edm.Decimal)
```

~~If implemented the~~The `floor` ~~canonical~~ function ~~MUST round~~rounds the input numeric parameter down to the nearest numeric value with no decimal component. The `floorMethodCallExpr` syntax rule defines how the `floor` function is invoked.

~~For example:~~

```
~~http://services.odata.org/Northwind/Northwind.svc/Orders?$filter=floor(Freight) eq 32~~
```

~~Returns~~Example 72: all orders ~~that have a~~with freight costs that round~~s~~ down to 32~~.~~

```
http://host/service/Orders?$filter=floor(Freight) eq 32
```

5.1.1.4.27 ceiling

The `ceiling` ~~canonical~~ function has the following signatures

```
Edm.Double ceiling(Edm.Double)
Edm.Decimal ceiling(Edm.Decimal)
```

~~If implemented the~~The `ceiling` ~~canonical~~ function ~~MUST round~~rounds the input numeric parameter up to the nearest numeric value with no decimal component. The `ceilingMethodCallExpr` syntax rule defines how the `ceiling` function is invoked.

~~For example:~~

```
~~http://services.odata.org/Northwind/Northwind.svc/Orders?$filter=ceiling(Freight) eq 32~~
```

~~Returns~~Example 73: all orders ~~that have a~~with freight costs that round~~s~~ up to 32~~.~~

```
http://host/service/Orders?$filter=ceiling(Freight) eq 32
```

5.1.1.4.28 isof

The `isof` ~~canonical~~ function has the following signatures

```
Edm.Boolean isof(type)
Edm.Boolean isof(expression,type)
```

~~If implemented the~~The single parameter `isof` ~~canonical~~ function ~~MUST return~~returns `true` if~~, and only if,~~ the current instance is assignable to the type specified~~. If implemented the~~, otherwise it returns `false`. The two parameter `isof` ~~canonical~~ function ~~MUST~~ return `true` if~~, and only if,~~ the object referred to by the expression is assignable to the type specified, otherwise it returns `false`.

The `isofExpr` syntax rule defines how the `isof` function is invoked.

~~For example:~~

Example 74: orders that are also `BigOrders`

```
http://~~services.odata.org/Northwind/Northwind.svc~~host/service/Orders?$filter=isof(NorthwindModel.BigOrder)
```

~~Returns only~~Example 75: orders of a customer that ~~are also `BigOrders`.~~is a `VIPCustomer`

```
http://~~services.odata.org/Northwind/Northwind.svc~~host/service/Orders?$filter=isof(Customer,NorthwindModel.~~M~~VIPCustomer)
```

~~Returns only orders that have a customer that is a MVPCustomer.~~

5.1.1.4.29 cast

The cast ~~canonical~~ function has the following signatures:

```
Edm.Any cast(type)
Edm.Any cast(expression,type)
```

~~If implemented the~~ The single parameter cast ~~canonical~~ function ~~MUST return~~returns the current instance cast to the type specified. ~~If implemented the~~The two-parameter cast ~~canonical~~ function ~~MUST return~~returns the object referred to by the expression cast to the type specified. ~~In either case if the~~

The cast function follows these rules:

- The null value can be cast to any type.
- Primitive types are cast to Edm.String or a type definition based on it by using the literal representation used in payloads, and WKT for Geo types. The cast fails if the target type specifies an insufficient MaxLength.
- Numeric primitive types are cast to each other with appropriate rounding. The cast fails if the integer part doesn't fit into target type.
- Edm.DateTimeOffset, Edm.Duration, and Edm.TimeOfDay values can be cast to the ~~canonical function MUST return NULL~~same type with a different precision with appropriate rounding.
- Collections are cast item by item.
- Services MAY support structural casting of entities and complex type instances to a derived type, or arbitrary structured type, by assigning values of identically named properties and casting them recursively. The cast fails if one of the property-value casts fails or the target type contains non-nullable properties that have not been assigned a value.

The cast function is optional for primitive values (first three rules).

If the cast fails the cast function returns null.

5.1.1.4.30 geo.distance

The geo.distance ~~canonical~~ function has the following signatures:

```
Edm.Double geo.distance(Edm.GeographyPoint,Edm.GeographyPoint)
Edm.Double geo.distance(Edm.GeometryPoint,Edm.GeometryPoint)

~~Edm.Double~~The geo.distance~~(Edm.GeometryPoint,Edm.GeometryPoint)~~
```

~~If implemented the geo.distance canonical~~ function ~~MUST return~~returns the shortest distance between the two points in the coordinate reference system signified by the two points' SRIDs.

5.1.1.4.31 geo.intersects

The geo.intersects ~~canonical~~ function has the following signatures:

```
Edm.Boolean geo.intersects(Edm.GeographyPoint,Edm.GeographyPolygon)
Edm.Boolean geo.intersects(Edm.GeometryPoint,Edm.GeometryPolygon)

~~Edm.Boolean~~The geo.intersects~~(Edm.GeometryPoint,Edm.GeometryPolygon)~~
```

~~If implemented the geo.intersects canonical~~ function ~~MUST return~~returns true if~~, and only if,~~ the specified point lies within the interior or on the boundary of the specified polygon, otherwise it returns false.

5.1.1.4.32 `geo.length`

The `geo.length` ~~canonical~~ function has the following signatures:

```
Edm.Double geo.length(Edm.GeographyLineString)
Edm.Double geo.length(Edm.GeometryLineString)
```

~~If implemented the~~The `geo.length` ~~canonical~~ function ~~MUST return~~returns the total length of its line string parameter in the coordinate reference system signified by its SRID.

5.1.1.5 Lambda Operators

OData defines two operators that evaluate a Boolean expression on a collection. Both must be prepended with a navigation path that identifies a collection. The argument of a lambda operator is a lambda variable name followed by a colon (`:`) and a Boolean expression that uses the lambda variable name to refer to properties of the related entities identified by the navigation path.

5.1.1.5.1 `any`

The `any` operator applies a Boolean expression to each member of a collection and ~~evaluates to~~returns `true` if ~~and only if~~ the expression is `true` for any member of the collection~~. As a special case the Boolean expression may be empty, in which case the~~, otherwise it returns `false`. The `any` operator ~~evaluates to~~ without an argument returns `true` if the collection is not empty.

~~For example:~~

~~`http://services.odata.org/Northwind/Northwind.svc/Orders?$filter=Order_Details/any(d:d/Quantity gt 100)`~~

~~returns~~ Example 76: all `Orders` that have any ~~`Orderlines`~~`Items` with a `Quantity` greater than `100`~~.~~

```
http://host/service/Orders?$filter=Items/any(d:d/Quantity gt 100)
```

5.1.1.5.2 `all`

The `all` operator applies a Boolean expression to each member of a collection and ~~evaluates to~~returns `true` if ~~and only if~~ the expression is `true` for all members of the collection, otherwise it returns `false`.

~~For example:~~

~~`http://services.odata.org/Northwind/Northwind.svc/Orders?$filter=Order_Details/all(d:d/Quantity gt 100)`~~

~~returns~~ Example 77: all `Orders` that have only ~~`Orderlines`~~`Items` with a `Quantity` greater than `100`~~.~~

```
http://host/service/Orders?$filter=Items/all(d:d/Quantity gt 100)
```

5.1.1.6 Literals

5.1.1.6.1 Primitive Literals

Primitive literals can appear in the resource path as key property values, and in the query part, for example, as operands in `$filter` expressions. They are represented according to the `primitiveLiteral` rule in **[OData-ABNF]**.

Example 78: Date literal

```
http://host/service/Products?$filter=ReleaseDate gt date'2013-05-24'
```

5.1.1.6.2 Complex and Collection Literals

Complex literals and collection literals in URLs are represented as JSON objects and arrays according to the `arrayOrObject` rule in **[OData-ABNF]**.

Note that the special characters `{`, `}`, `[`, `]`, and `"` MUST be percent-encoded in URLs although some browsers will accept and pass them on unencoded.

Example 79: collection of string literals

```
http://host/service/ProductsByColor?colors=["red","green"]
```

5.1.1.6.3 `null`

The `null` literal can be used to compare a value to null, or to pass a null value to a function.

5.1.1.6.4 `$it`

The `$it` literal can be used in expressions to refer to the current instance of the collection identified by the resource path. It can be used to compare properties of related entities to properties of the current instance in expressions within lambda operators, for example in `$filter` and `$orderby` expressions on collections of primitive types, or in `$filter` expressions nested within `$expand`.

Example 80: email addresses ending with `.com` assuming `EmailAddresses` is a collection of strings

```
http://host/service/Customers(1)/EmailAddresses?$filter=endswith($it,'.com')
```

Example 81: customers along with their orders that shipped to the same city as the customer's address. The nested filter expression is evaluated in the context of Orders; `$it` allows referring to values in the outer context of Customers.

```
http://host/service/Customers?
            $expand=Orders($filter=$it/Address/City eq ShipTo/City)
```

5.1.1.6.5 `$root`

The `$root` literal can be used in expressions to refer to resources of the same service. It can be used as a single-valued expression or within complex or collection literals.

Example 82: all employees with the same last name as employee `A1235`

```
http://host/service/Employees?
                         $filter=LastName eq $root/Employees('A1245')/LastName
```

Example 83: products ordered by a set of customers, where the set of customers is passed as a JSON array containing the resource paths from `$root` to each customer.

```
http://host/service/ProductsOrderedBy(Customers=@c)?
                        @c=[$root/Customers('ALFKI'),$root/Customers('BLAUS')]
```

~~5.1.1.6~~5.1.1.7 Path Expressions

Properties and navigation properties of the entity type of the set of resources that are addressed by the request URL can be used as operands or function parameters, as shown in the preceding examples.

Properties of complex properties can be used via the same syntax as in resource paths, i.e. by specifying the name of a complex property, followed by a forward slash `/` and the name of a property of the complex property, and so on,

Properties and navigation properties of entities related with a target cardinality 0..1 or 1 can be used by specifying the navigation property, followed by a forward slash `/` and the name of a property of the related entity, and so on.

If a complex property is `null`, or no entity is related (in case of target cardinality 0..1), its value, and the values of its components, are treated as `null`.

~~For example, the path expression:~~

Example 84: similar behavior whether `HeadquarterAddress` is a nullable complex type or a nullable navigation property

```
Companies(1)/HeadquarterAddress/Street
```

~~will show similar behavior if modeled with a nullable property `HeadquarterAddress` of complex type `Address` or an optional navigation property `HeadquarterAddress` targeting an `Address` entity with some artificial key.~~

Properties of derived types can be used by specifying the qualified name of a derived type, followed by a forward slash `/` and the name of the property of the derived type, see ~~addressing derived types.~~addressing derived types. If the current instance is not of the specified derived type, the path expression ~~evaluates to~~returns `null`.

~~5.1.1.7~~5.1.1.8 Parameter Aliases

~~Expressions~~Parameter aliases can be used within `$filter` or `$orderby` in place of expressions that evaluate to a primitive value, a complex value, or a collection of primitive or complex values~~ may be “outsourced” to a separate query option that starts~~. Parameter names start with ~~an @~~the at sign~~,~~ (`@`) and ~~the name of this query option~~ can be used in ~~one or~~ more ~~places~~than one place in the expression. The value for the parameter alias is supplied in a query option with the same name as the parameter.

~~For example:~~Example 85:

```
http://host/service/Movies?$filter=~~substringof~~contains(@word,Title)&@word='Black'
```

~~or even~~Example 86:

```
http://host/service/Movies?$filter=Title eq @title&@title='Wizard of Oz'
```

~~5.1.1.8~~5.1.1.9 Operator Precedence

OData services MUST use the following operator precedence for supported operators when evaluating ~~`$filter`~~`$filter` and ~~`$orderby`~~`$orderby` expressions. Operators are listed by category in order of precedence from highest to lowest. Operators in the same category have equal precedence:

Group	Operator	Description	ABNF Expression
Grouping	`( )`	Precedence grouping	`parenExpr` `boolParenExpr`
Primary	`/`	Navigation	`firstMemberExpr` `memberExpr`

	`xxx( )`	Method Call	`methodCallExpr`
			`boolMethodCallExpr`
			`functionExpr`

Unary	`-`	Negation	`negateExpr`
	~~`n`~~`Not`	Logical Negation	`notExpr`
	`cast( )`	Type Casting	`castExpr`

Group	Operator	Description	ABNF Expression
Multiplicative	~~m~~Mul	Multiplication	mulExpr
	~~d~~Div	Division	divExpr
	~~m~~Mod	Modulo	modExpr
Additive	~~a~~Add	Addition	addExpr
	~~s~~Sub	Subtraction	subExpr
Relational	~~gt~~Gt	Greater Than	gtExpr
	~~ge~~Ge	Greater than or Equal	geExpr
	~~lt~~Lt	Less Than	ltExpr
	~~le~~Le	Less than or Equal	leExpr
	isof	Type Testing	isofExpr
Equality	~~eq~~Eq	Equal	eqExpr
	~~ne~~Ne	Not Equal	neExpr
Conditional AND	~~a~~And	Logical And	andExpr
Conditional OR	~~or~~Or	Logical Or	orExpr

~~5.1.1.9~~5.1.1.10 Numeric Promotion

Services MAY support numeric promotion for arithmetic operations or when comparing two operands of comparable types by applying the following rules, in order:

1. If either operand is `Edm.Double`, the other operand is converted to type `Edm.Double`.
2. Otherwise, if either operand is `Edm.Single`, the other operand is converted to type `Edm.Single`.
3. Otherwise, if either operand is of type `Edm.Decimal`, the other operand is converted to `Edm.Decimal`.
4. Otherwise, if either operand is `Edm.Int64`, the other operand is converted to type `Edm.Int64`.
5. Otherwise, if either operand is `Edm.Int32`, the other operand is converted to type `Edm.Int32`
6. Otherwise, if either operand is `Edm.Int16`, the other operand is converted to type `Edm.Int16`.

~~For each~~Each of these promotions~~, a service SHOULD use~~ uses the same semantics as a `castExpression` to promote an operand to the target type.

If the result of an arithmetic operation does not fit into the type determined by the above rules, the next-wider type is used in the above order, with `Edm.Double` considered widest.

OData does not define an implicit conversion between string and numeric types.

5.1.2 ~~Expand~~ System Query Option `$expand`

The `$expand` system query option ~~allows clients to request~~ specifies the related resources ~~when a resource that satisfies a particular request is~~to be included in line with retrieved resources.

What follows is a (non-normative) snippet from **[OData-ABNF]** that ~~applies to the Expand System Query Option~~describes the syntax of `$expand`:

```
expand           = '$expand' EQ expandItem *( COMMA expandItem )

expandItem ~~= [ qualifiedEntityTypeName "/" ]~~
```

```
                  ~~*( ( complexProperty / complexColProperty ) "/"~~
                      ~~[ qualifiedComplexTypeName "/" ] ) navigationProperty~~
                = STAR [ ref ~~[ "("~~/ OPEN levels CLOSE ]
                         / expandPath
                            [ ref     [ OPEN expandRefOption
                                     *( SEMI expandRefOption    ) ~~")" ]~~
CLOSE ]
                            / count [ ~~"("~~OPEN expandCountOption
                                     *( SEMI expandCountOption ) ~~")" ]~~
                  ~~/~~            ~~"("~~CLOSE ]
                            /         OPEN expandOption
                                     *( SEMI expandOption ~~")"~~      ) CLOSE
                            ]
                  ~~]~~

expandPath         = [ qualifiedEntityTypeName "/" ]
                     *( ( complexProperty / complexColProperty ) "/"
                        [ qualifiedComplexTypeName "/" ] )
                     navigationProperty
                     [ "/" qualifiedEntityTypeName ]

expandCountOption = filter
                   / search

expandRefOption    = expandCountOption
                   / orderby
                   / skip
                   / top
                   / inlinecount

expandOption       = expandRefOption
                   / select
                   / expand
                   / levels
```

Each `expandItem` ~~MUST be~~is evaluated relative to the entity containing the navigation property being expanded~~ entity~~.

A type cast using the `qualifiedEntityTypeName` to a type containing the property is required in order to expand a navigation property defined on a derived type.

An arbitrary number of single- or collection-valued complex properties, optionally followed by a type cast, allow~~s~~ drilling into complex proper~~it~~ies.

The `navigationProperty` segment MUST identify a navigation property defined on the entity type of the request, the derived entity type specified in the type cast, or the last complex type identified by the complex property path.

```
~~http://services.odata.org/OData/OData.svc/Products?$expand=Category~~
```

Example 87: expand a navigation property of an entity type

```
http://host/service/Products?$expand=Category
```

Example 88: expand a navigation property of a complex type

```
http://host/service/Customers?$expand=Addresses/Country
```

A navigation property MUST NOT appear in more than one `expandItem`.

~~The~~Query options can be applied to the expanded navigation property ~~name MAY be followed by~~by appending a semicolon-separated list of ~~system ~~query options, enclosed in parentheses~~. These are evaluated on the entities identified by~~, to the navigation property~~:~~ name.

```
http://~~services.odata.org/OData/OData.svc~~host/service/Categories~~?$~~?
                                          $expand=Products($filter=DiscontinuedDate eq null)
```

~~Allowed system query options are `$filter`, `$select`, `$orderby`, `$skip`, `$top`, `$count`, `$search`, and `$expand` (optionally followed by another list of nested options).~~

Allowed system query options are `$filter`, `$select`, `$orderby`, `$skip`, `$top`, `$count`, `$search`, and `$expand`.

To retrieve entity references instead of the related entities, append `/$ref` to the navigation property name or ~~type segment~~type-cast segment following a navigation property name~~:~~.

Example 89: all categories and for each category the references of all related products

```
http://~~services.odata.org/OData/OData.svc~~host/service/Categories?$expand=Produ
cts/$ref
```

~~returns categories and, for each category, the references of all related products for that category.~~

```
~~http://services.odata.org/OData/OData.svc/Categories?$expand=Products/Sales.Pr
emierProduct/$ref~~
```

~~returns ~~Example 90: all categories and~~,~~ for each category~~,~~ the references of all related products ~~for that category that are ~~of the derived type `Sales.PremierProduct`~~.~~

```
http://~~services.odata.org/OData/OData.svc~~host/service/Categories?$expand=Produ
cts/Sales.PremierProduct/$ref~~($filter=CurrentPromotion eq null)~~
```

~~returns~~Example 91: all categories and~~,~~ for each category~~,~~ the references of all related premier products ~~for that category that have ~~with a current promotion equal to `null`~~.~~

```
http://host/service/Categories?
  $expand=Products/Sales.PremierProduct/$ref($filter=CurrentPromotion eq null)
```

Cyclic navigation properties (whose target type is identical or can be cast to its source type) ~~MAY additionally specify~~can be recursively expanded using the special ~~option ~~`$levels`~~, followed by an equals-sign and~~ option. The value of the `$levels` option is either a positive integer to specify the number of levels to expand, or the literal string `max`~~. In this case the navigation property is recursively expanded up to the specified level, with `max` meaning~~ to specify the maximum expansion level supported by that service~~:~~.

Example 92: all employees with their manager, manager's manager, and manager's manager's manager

```
http://~~contoso.com/HR~~host/service/Employees?$expand=Model.Manager/DirectReport
s($levels=~~4~~3)
```

~~Select ~~It is also possible to expand all declared and dynamic navigation properties using a star (`*`). To retrieve references to all related entities use `*/$ref`, and to expand all related entities with a certain distance use the star operator with the `$levels` option. The star operator can be combined with explicitly named navigation properties, which take precedence over the star operator.

Example 93: expand `Supplier` and include references for all other related entities

```
http://host/service/Categories?$expand=*/$ref,Supplier
```

Example 94: expand all related entities and their related entities

```
http://host/service/Categories?$expand=*($levels=2)
```

5.1.3 System Query Option $select

The `$select` system query option allows clients to requests a limited set of ~~information~~properties for each entity or complex type~~ identified by the `resourcePath` and other System Query Options like `$filter`, `$top`, `$skip` etc. ~~.

The `$select` query option is often used in conjunction with the ~~expand system query option, to first increase the scope~~`$expand` system query option, to define the extent of the resource graph ~~returned~~to return (`$expand`) and then ~~selectively prune that~~specify a subset of properties for each resource in the graph (`$select`). Expanded navigation properties MUST be returned, even if they are not specified as a `selectItem`.

What follows is a (non-normative) snippet ~~**[OData-ABNF]**~~from **[OData-ABNF]** ~~that applies to the Select System Query Option~~showing the syntax of `$select`:

```
select          = '$select' EQ selectItem *( COMMA selectItem )

selectItem     = STAR
               / allOperationsInSchema
             ~~/[~~
               / [ qualifiedEntityTypeName "/" ]
             ~~( *( ( complexProperty / complexColProperty ) "/"~~
                ~~[ qualifiedComplexTypeName "/" ]~~

                ~~) ( property  / navigationProperty )~~
     ( selectProperty
                 / qualifiedActionName
                 / qualifiedFunctionName
                 )

selectProperty = primitiveProperty
               / primitiveColProperty
               / navigationProperty
               / selectPath [ "/" selectProperty ]

selectPath     = ( complexProperty / complexColProperty )
                 [ "/" qualifiedComplexTypeName ]
```

The `$select` system query option ~~MUST be~~is interpreted relative to the entity type or complex type of the resources identified by the resource path section of the URL~~, for example:~~

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

~~In this URL the "`Rating,ReleaseDate`" clause MUST be interpreted relative to the `Product` entity type which is the entity type of the resources identified by this~~

```
~~http://services.odata.org/OData/OData.svc/Products~~
```

~~URL.~~

. Each `selectItem` in the `$select` clause indicates that the response MUST include the declared or dynamic properties, actions and functions identified by that `selectItem`. The simplest form of a `selectItem` explicitly requests a property defined on the entity type of the resources identified by the resource path section of the URL~~, for example this URL requests just the `Rating` and `ReleaseDate` for the matching Products:~~.

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

Example 95: rating and release date of all products

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

It is also possible to request all declared and dynamic structural properties~~,~~ using a star ~~(*) request:~~(*).

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

~~If the selectItem is a star, then~~ Example 96: all structural properties of ~~the matching resources MUST be returned.~~all products

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

If the selectItem is a navigation property ~~that does not appear~~ then the corresponding navigation link is represented in ~~an $expand~~the response. If the navigation property also appears in an $expand query option~~, the navigation property MUST be~~ then it is additionally represented as ~~deferred~~inline content. ~~If it also appears in an $expand query option, it is represented as inlined~~This inline content~~. This inlined content may~~ can itself be restricted with a nested $select query option, see section 5.1.1.10. ~~For example:~~

Example 97: name and description of all products, plus name of expanded category

```
http://~~services.odata.org/OData/OData.svc~~host/service/Products~~?$~~?

$select=Name,Description~~,Category~~&$expand=Category($select=Name)
```

~~will return the Name, Description, and Category properties of~~Qualifying the selectItem with a qualifiedEntityTypeName prefix casts the ~~Product~~ entity ~~type,~~ or complex value to the ~~latter as inlined content containing only the Name~~ specified type in order to select a property of ~~the Category entity~~that derived type.

A selectItem ~~MAY include a cast to a derived type using a qualifiedEntityTypeName prefix.~~

~~If the property in the selectItem~~that is ~~of~~a complex type or collection of complex type~~, it MAY~~ can be followed by a forward slash, an~~d~~ optional type cast segment, and the name of a property of the complex type (and so on for nested complex types). ~~For example the following URL requests~~

Example 98: the AccountRepresentative property of any supplier that is of the derived type Namespace.PreferredSupplier~~ is selected~~, together with the Street property of the complex property Address, and the Location property of the derived complex type Namespace.AddressWithLocation~~:~~

```
http://host/service~~.odata.org/OData/OData.svc~~/Suppliers~~?$~~?
     $select=Namespace.PreferredSupplier/AccountRepresentative,
             Address/Street,
             Address/Namespace.AddressWithLocation/Location
```

~~If a~~Any structural property, ~~open property,~~ non-expanded navigation property, or operation ~~is~~ not requested as a selectItem (explicitly or via a star~~), it~~) SHOULD ~~NOT~~ be ~~included in~~omitted from the response.

~~A star SHOULD NOT reintroduce actions or functions. Thus if~~ If any selectItem (including a star) is specified, actions and functions SHOULD be omitted unless explicitly requested using a qualifiedActionName, a qualifiedFunctionName or the allOperationsInSchema~~ clause~~.

~~Actions and Functions information can be explicitly~~ If an action or function is requested ~~with~~as a selectItem~~ containing~~, either ~~a~~ explicitly by using a qualifiedActionName or ~~a~~ qualifiedFunctionName cause, or ~~can be~~ implicitly ~~requested~~by using ~~a selectItem contain the~~ allOperationsInSchema~~ clause.~~

~~For example this~~, then the service includes information about how to invoke that operation for each entity identified by the last path segment in the request URL ~~requests~~for which the operation can be bound.

If an action or function is requested in a selectItem using a qualifiedActionName or a qualifiedFunctionName and that operation cannot be bound to the entities requested, the service MUST ignore the selectItem.

Example 99~~:~~: the `ID` property, the `ActionName` action defined in `Model` and all actions and functions defined in the `Model2` for each product~~,~~ if those actions and functions can be bound to that product~~:~~

```
http://~~services.odata.org/OData/OData.svc~~host/service/Products?$select=ID,Mode
l.ActionName,Model2.*
```

~~If an action is requested as a `selectItem`, either explicitly by using a `qualifiedActionName` clause or implicitly by using an `allOperationsInSchema` clause, then for each entity identified by the last path segment in the request URL for which the action can be bound the service MUST include information about how to invoke that action.~~

~~If a function is requested as a `selectItem`, either explicitly by using an `qualifiedFunctionName` clause or implicitly by using an `allOperationsInSchema` clause, the service MUST include in the response information about how to invoke that function for each of the entities that are identified by the last path segment in the request URL, if and only if the function can be bound to those entities.~~

~~If an action or function is requested in a `selectItem` using a `qualifiedActionName` or a `qualifiedFunctionName` clause and that action or function cannot be bound to the entities requested, the service MUST ignore the `selectItem` clause.~~

When multiple `selectItems` exist in a select clause, then the total set of propert~~y~~ies, open propert~~y~~ies, navigation propert~~y~~ies, actions and functions to be returned is equal to the union of the set of those identified by each `selectItem`.

If a `selectItem` is a path expression requesting a component of a complex property and the complex property is `null` on an instance, then the component is treated as `null` as well.

~~Redundant `selectItems` on the same URL MAY be considered valid, but MUST NOT alter the meaning of the URL.~~

5.1.4 ~~OrderBy~~ System Query Option $orderby

The `$orderby` system query option allows clients to request ~~a resource~~resources in a particular order.

The semantics of `$orderby` are covered in the **[OData-Protocol]** document.

The **[OData-ABNF]** `orderby` syntax rule defines the formal grammar of the `$orderby` query option.

5.1.5 ~~Top and Skip~~ System Query Options $top and $skip

The `$top` system query option ~~allows clients a required~~ requests the number of ~~resources, used~~items in ~~conjunction~~the queried collection to be included in the result. The `$skip` query option ~~which allows a client to ask the service to begin sending resources after skipping a required~~requests the number of ~~resources, a~~items in the queried collection that are to be skipped and not included in the result. A client can request a particular page of ~~matching resources~~items by combining `$top` and `$skip`.

The semantics of `$top` and `$skip` are covered in the ~~document.~~

[OData-Protocol] document. The **[OData-ABNF]** `top` and `skip` syntax rules define the formal grammar of the `$top` and `$skip` query options respectively.

5.1.6 ~~Count~~ System Query Option $count

The `$count` system query option allows clients to request a count of the ~~number of~~ matching resources inc~~l~~in~~ud~~ed with the resources in the response. ~~Typically this is most useful when~~The `$count` query option has a ~~service implements server-side paging, as it allows clients to retrieve the number of matching resources even if the service decides to only respond with a single page of matching resources~~Boolean value of `true` or `false`.

The semantics of `$count` is covered in the ~~**[OData-Protocol]**~~**[OData-Protocol]** document.

~~The `$count` query option takes one of the Boolean values `true` or `false` as its argument.~~

5.1.7 ~~Search~~ System Query Option $search

The `$search` system query option allows clients to request entities matching a free-text ~~search expression~~search expression.

The `$search` ~~system~~ query option can be applied ~~directly to a service URL in order to return all matching entities within the service, to a URL representing an entity container in order to return all matching entities within any entity sets contained by the entity container, or~~ to a URL representing a collection of entities to return all matching entities within the collection.

~~If both a~~ Applying the `$search` query option ~~and a `$filter` query option~~ to the `$all` resource requests all matching entities in the service.

If both `$search` and `$filter` are applied to the same request, the results include only those entities that match both criteria.

The **[OData-ABNF]** `search` syntax rule defines the formal grammar of the `$search` query option.

~~For example~~

~~`http://host/service/Products?$search=blue OR green`~~

~~*searches for*~~*Example 100: all products that are blue or green. It is up to the service to decide what makes a product blue or green.*

```
http://host/service/Products?$search=blue OR green
```

5.1.7.1 Search Expressions

Search expressions are used within the ~~`$search` system query option~~`$search` system query option to request entities matching the specified expression.

Terms ~~may~~can be any single word to be matched within the expression.

Terms enclosed in double-quotes comprise a *phrase*.

Each individual term or phrase comprises a ~~b~~Boolean expression that ~~evaluates to~~returns `true` if~~, and only if,~~ the term or phrase is matched, otherwise `false`. The semantics of what is considered a match is dependent upon the service.

Expressions enclosed in parenthesis comprise a *group expression*.

The search expression ~~MAY~~can contain any number of terms, phrases, or group expressions, along with the case-sensitive keywords `NOT`, `AND`, and `OR`, evaluated in that order.

Expressions prefaced with `NOT` evaluate to `true` if~~, and only if,~~ the expression is not matched, otherwise `false`.

Two expressions not enclosed in quotes and separated by a space are equivalent to the same two expressions separated by the `AND` keyword. Such expressions evaluate to `true` if~~, and only if,~~ both of the expressions evaluate to `true`, otherwise `false`.

Expressions separated by an `OR` evaluate to `true` if~~, and only if,~~ either of the expressions evaluate to `true`, otherwise `false`.

The **[OData-ABNF]** `searchExpr` syntax rule defines the formal grammar of the search expression.

5.1.8 ~~Format~~ System Query Option $format

The `$format` system query option allows clients to request a response in a particular format~~. Generally requesting a particular format~~ and is ~~done using~~useful for clients without access to request headers for standard content-type negotiation~~, however occasionally the client has no access to request headers which makes standard content type negotiation not an option, it is in these situations that `$format` is generally used.~~. Where present `$format` takes precedence over standard content-type negotiation.

The semantics of `$format` is covered in the ~~**[OData-Protocol]**~~**[OData-Protocol]** document.

The **[OData-ABNF]** `format` syntax rule define the formal grammar of the `$format` query option.

5.2 Custom Query Options

Custom query options provide an extensible mechanism for ~~data~~ service-specific information to be placed in a ~~data service~~ URL query string. A custom query option is any query option of the form shown by the rule `customQueryOption` in **[OData-ABNF]**.

Custom query options MUST NOT begin with a ~~“$”~~`$` or ~~“@”~~`@` character.

~~For example this URL addresses provide a `securitytoken` via a~~Example 101: service-specific custom query option~~:~~ `debug-mode`

```
http://~~services.odata.org/OData/OData.svc~~host/service/Products?~~securitytoken=0
412312321~~debug-mode=true
```

~~5.3 URL Equivalence~~

~~When determining if two URLs are equivalent, each URL SHOULD be normalized using the rules specified in **[RFC3986]** and **[RFC3987]** and then compared for equality using the equivalence rules specified in **[RFC2616]**, Section 3.2.3.~~

5.3 Parameter Aliases

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

Parameter aliases MUST start with an `@` character.

The semantics of function parameter aliases are covered in **[OData-Protocol]**.

The **[OData-ABNF]** rule `aliasAndValue` defines the formal grammar for passing function parameter aliases as query options.

6 Conformance

The conformance requirements for OData clients and services are described in ~~[OData-Protocol]~~**[OData-Protocol]**.

Appendix A. Acknowledgments

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

Appendix B. Revision History

Revision	Date	Editor	Changes Made
Working Draft 01	2012-08-22	Michael Pizzo	Translated Contribution to OASIS format/template
Committee Specification Draft 01	2013-04-26	Ralf Handl Michael Pizzo Martin Zurmuehl	Added FullText Search, modified expand syntax, expand options, crosstabs, enumerations Fleshed out descriptions and examples and addressed numerous editorial and technical issues processed through the TC Added Conformance section
Committee Specification Draft 02	2013-07-01	Ralf Handl Michael Pizzo Martin Zurmuehl	Described which query options are applicable to which resource types and HTTP methods Simplified URL syntax Extended expand with a STAR operator Added special resources for cross-service search, cross joins, resolution of entity-ids Described handling of null values, division by zero, and overflow in arithmetic operations Added filtering for collections of complex and primitive types