URL Conventions (OData Version 3.0)

OData Version 4.0 is the current recommended version of OData. OData V4 has been standardized by OASIS and has many features not included in OData Version 3.0.

□ **Go to OData Version 4.0 (/documentation/)**

1. Introduction

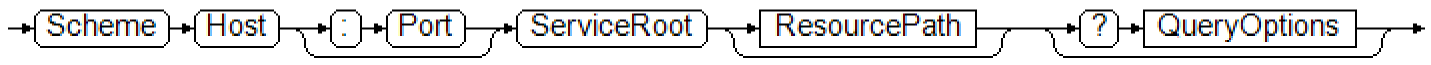
The Open Data Protocol (OData) enables the creation of REST-based data services, which allow resources, identified using Uniform Resource Identifiers (URLs) and defined in a data model, to be published and edited by Web clients using simple HTTP messages. This specification defines a set of recommended (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] documents specify the format of the resource representations that are exchanged using OData and the [OData:Operations] document describes the actions that can be performed on the URLs (optionally constructed following the conventions defined in this document) embedded in those representations.

Servers 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.

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 be present in a URL used by an OData service; however, this specification applies no further meaning to such additional constructs.



(/assets/ODataUri.png)

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

```
https://services.odata.org/OData/OData.svc
_______________________________________/
                   |
         service root URL

https://services.odata.org/OData/OData.svc/Category(1)/Products?$top=2&$orderby=name
_______________________________________/ __________________/ __________________/
                   |                                |                    |
          service root URL                  resource path        query options
```

3. Service Root URL

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

4. Resource Path

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

The resource path section of a URL identifies the resource to be interacted with (such as Customers, a single Customer, Orders related to Customers in London, and so forth). The resource path enables any aspect of the data model (collections of entities, a single entity, properties, Links, service operations, and so on) exposed by an OData service to be addressed.

4.1. Addressing entities

The basic rules for addressing a collection (of entities), a single entity within a collection, as well as a property of an entity are covered in the 'resourcePath' syntax rule in Appendix A.

Below is a snippet from Appendix A:

```
resourcePath                  =   [ entityContainerName "." ] entitySetName [collectionNavigation] /
                                  ( entityColServiceOpCall / entityColFunctionCall ) [ collectionNavigation ] /
                                  ( entityServiceOpCall   / entityFunctionCall ) [ singleNavigation ] /
                                  ( complexColServiceOpCall / complexColFunctionCall ) [ boundOperation ] /
                                  ( complexServiceOpCall / complexFunctionCall ) [ boundOperation / complexpropertyPath ] /
                                  ( primitiveColServiceOpCall / primitiveColFunctionCall ) [ boundOperation ] /
                                  ( primitiveServiceOpCall / primitiveFunctionCall ) [ boundOperation / value ] /
                                  actionCall
```

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)

  ```
  For example: https://services.odata.org/OData/OData.svc/Products
  ```

- By invoking a function that returns a collection of entities (see rule: entityColFunctionCall)

  ```
  For example: https://services.odata.org/OData/OData.svc/GetProductsByCategoryId(categoryId=2)
  ```

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

  ```
  For example: https://services.odata.org/OData/OData.svc/ProductsByColor?color='red'
  ```

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)
- Invoking a service operation that returns a single entity (see rule: entityServiceOpCall)

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 a entity key to select a single entity (see rules: collectionNavigation and keyPredicate)

  ```
  For example: https://services.odata.org/OData/OData.svc/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: https://services.odata.org/OData/OData.svc/Products/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: https://services.odata.org/OData/OData.svc/Products(1)/Supplier
  ```

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

  ```
  For example: https://services.odata.org/OData/OData.svc/Products(1)/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: https://services.odata.org/OData/OData.svc/Categories(1)/Products
  ```

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

  ```
  For example: https://services.odata.org/OData/OData.svc/Categories(1)/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: https://services.odata.org/OData/OData.svc/Categories(1)/Products/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 a entity or collections of entities.

4.1.1. 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 entitySet associated with the entity followed by the key predicate identifying the entity within the collection.

For example the URLs https://services.odata.org/OData/OData.svc/Categories(1)/Products(1) (https://services.odata.org/OData/OData.svc/Categories%281%29/Products%281%29) and https://services.odata.org/OData/OData.svc/Products(1) (https://services.odata.org/OData/OData.svc/Products%281%29) represent the same entity, but the canonical URL for the entity is https://services.odata.org/OData/OData.svc/Products(1) (https://services.odata.org/OData/OData.svc/Products%281%29).

For contained entities the canonical URL begins with canonical URL of the parent, with further path segments that:

- Name and navigate through the containing navigation property
- and, if the navigation property returns a collection, an entity key (see rule: entity key) that uniquely identifies the entity in that collection.

4.2. Addressing Links between Entities

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

Links between entities are addressable in OData just like entities themselves are (as described above). The basic rules for addressing relationships are shown in the following figure. By the following rule:

```
entityURL       =   ; any URL that identifies a single entity
                    ; examples include: an entitySet followed by a key or a function/serviceOperation that returns a single entit
y.
links           =   entityURL "/$links/" navigationpropertyName
```

For example: https://services.odata.org/OData/OData.svc/Category(1)/$links/Products addresses the links between Category(1) and Products.

4.3. Addressing Operations

4.3.1. Addressing Actions

The semantic rules for addressing and invoking actions are defined in the OData:Core (../odata-version-3-0-core-protocol) document. The grammar for addressing and invoking actions are defined by the following syntax grammar rules in Appendix A:

- The actionCall syntax rule defines the grammar in the ResourcePath for addressing and invoking an action 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 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.

4.3.2. Addressing Functions

The semantic rules for addressing and invoking functions are defined in the OData:Core (../odata-version-3-0-core-protocol) document. The grammar for addressing and invoking functions is defined by a number syntax grammar rules in Appendix A, in particular:

- The functionCall syntax rule defines the grammar in the ResourcePath for addressing and providing parameters for a function directly from the Service Root.
- The boundFunctionCall syntax rule defines 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 be used as the binding parameter value when invoking the function.
- The boundOperation syntax rule (which encompasses the boundFunctionCall syntax rule), when used by the resourcePath syntax rule, illustrates how a boundFunctionCall can be appended to a ResourcePath.
- The functionExpr, boolFunctionExpr, boundFunctionExpr and boolBoundFunctionExpr 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:Core 10.4.2.3.2 (../odata-version-3-0-core-protocol#parameteraliases).
- The parameterAndValue syntax rule defines the grammar for providing function parameter values using Parameter Name Syntax OData:Core 10.4.2.3.3 (../odata-version-3-0-core-protocol#parameternamesyntax).

4.3.3. Addressing Legacy Service Operations

The semantic rules for addressing and invoking a service operation are defined in the OData:Core (../odata-version-3-0-core-protocol) document. The grammar for addressing and invoking a service operation is define by 3 syntax rules in Appendix A:

- The serviceOperationCall syntax rule defines the grammar for the ResourcePath that addresses the service operation.
- The resourcePath syntax rule defines the grammar for any additional composition of OData ResourcePath segments that rely on the results of calling the service operation.
- The sopParameterNameAndValue syntax rule defines the grammar for specifying any parameters to the service operation in the query part of the request URL.

This example, illustrates a call to a service operation with subsequent resource path segments, that uses all three syntax rules: [https://services.odata.org/OData/OData.svc/GetProductsByRating?rating=3&$filter=Price gt 20.0M] (https://services.odata.org/OData/OData.svc/GetProductsByRating?rating=3&$filter=Price gt 20.0M)

This invokes the GetProductsByRating service operation, with the rating parameter value set to 3, and then subsequently filters Products returned by the service operation call to include only those with a price greater than $20.

4.4. Addressing a Property

To address an entity property clients compose the property name, to the url of the entity, in a new url segment. If the property has a complex type value, properties of that value can be addressed by further property name composition.

4.5. Addressing a Property Value

To address the raw value of a property, client compose `/$value` to the property url.

5. Query Options

The Query Options section of an OData URL specifies three types of information: System Query Options, Custom Query Options, and Operation (function and service operation) Parameters. 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 are query string parameters a client may specify to control the amount and order of the data that an OData service returns for the resource identified by the URL. The names of all System Query Options are prefixed with a “$” character.

An OData service may support some or all of the System Query Options defined. If a data service does not support a System Query Option, it must reject any requests which contain the unsupported option.

The semantics of all System Query Options are defined in the OData:Core (../odata-version-3-0-core-protocol) document.

The grammar and syntax rules the System Query Options are defined in

5.1.2. Filter System Query Option

The $filter system query option allows clients to filter the set of resources that are addressed by a request URL. $filter specifies conditions that MUST be met by a resource for it to be returned in the set of matching resources.

The semantics of $filter are covered in the OData:Core (../odata-version-3-0-core-protocol) document.

The OData:ABNF (../abnf) filter syntax rule defines the formal grammar of the $filter query option.

5.1.2.1. Logical Operators

OData defines a set of logical operators that evaluate to true or false (i.e. a boolCommonExpr as defined in Appendix A). Logical Operators are typically used in the Filter System Query Option to filter the set of resources. However Servers MAY allow for the use of Logical Operators with the OrderBy System Query Option.

The syntax rules for the Logical Operators are defined in Appendix A.

5.1.2.1.1. Equals Operator

The Equals operator (or ‘eq’) evaluates to true if the left operand is equal to the right operand, otherwise if evaluates to false.

5.1.2.1.2. Not Equals Operator

The Not Equals operator (or ‘ne’) evaluates to true if the left operand is not equal to the right operand, otherwise if evaluates to false.

5.1.2.1.3. Greater Than Operator

The Greater Than operator (or ‘gt’) evaluates to true if the left operand is greater than the right operand, otherwise if evaluates to false.

5.1.2.1.4. Greater Than or Equal Operator

The Greater Than or Equal operator (or ‘ge’) evaluates to true if the left operand is greater than or equal to the right operand, otherwise if evaluates to false.

5.1.2.1.5. Less Than Operator

The Less Than operator (or ‘lt’) evaluates to true if the left operand is less than the right operand, otherwise if evaluates to false.

5.1.2.1.6. Less Than or Equal Operator

The Less Than operator (or ‘le’) evaluates to true if the left operand is less than or equal to the right operand, otherwise if evaluates to false.

5.1.2.1.7. Logical And Operator

The Logical And operator (or ‘and’) evaluates to true if both the left and right operands both evaluate to true, otherwise if evaluates to false.

5.1.2.1.8. Logical Or Operator

The Logical Or operator (or ‘or’) evaluates to false if both the left and right operands both evaluate to false, otherwise if evaluates to true.

5.1.2.1.9. Logical Negation Operator

The Logical Negation Operator (or ‘not’) evaluates to true if the operand evaluates to false, otherwise it evalutes to false.

5.1.2.1.10. Logical Operator Examples

The following examples illustrate the use and semantics of each of the logical operators:

```
https://services.odata.org/OData/OData.svc/Products?$filter=Name eq 'Milk' (Requests all products with a Name equal to 'Milk').

https://services.odata.org/OData/OData.svc/Products?$filter=Name ne 'Milk' (Requests all products with a Name not equal to 'Mil
k').

https://services.odata.org/OData/OData.svc/Products?$filter=Name gt 'Milk' (Requests all products with a Name greater than 'Mil
k').

https://services.odata.org/OData/OData.svc/Products?$filter=Name ge 'Milk' (Requests all products with a Name greater than or equ
al to 'Milk').

https://services.odata.org/OData/OData.svc/Products?$filter=Name lt 'Milk' (Requests all products with a Name less than 'Milk').

https://services.odata.org/OData/OData.svc/Products?$filter=Name le 'Milk' (Requests all products with a Name less than or equal
to 'Milk').

https://services.odata.org/OData/OData.svc/Products?$filter=Name eq 'Milk' and Price lt '2.55M' (Requests all products with the N
ame 'Milk' that also have a Price less than 2.55).

https://services.odata.org/OData/OData.svc/Products?$filter=Name eq 'Milk' or Price lt '2.55M' (Requests all products that either
have the Name 'Milk' or have a Price less than 2.55).

https://services.odata.org/OData/OData.svc/Products?$filter=not endswith(Name, 'ilk') (Requests all products that do not have a N
ame that ends with 'ilk').
```

5.1.2.2. Arithmetic Operators

OData defines a set of arithmetic operators that require operands that evaluate to numeric types. Arithmetic Operators are typically used in the Filter System Query Option to filter the set of resources. However Servers MAY allow for the use of Arithmetic Operators with the OrderBy System Query Option.

The syntax rules for the Arithmetic Operators are defined in Appendix A.

5.1.2.2.1. Addition Operator

The Addition Operator (or ‘add’) adds the left and right numeric operands together.

5.1.2.2.2. Subtraction Operator

The Subtraction Operator (or ‘sub’) subtracts the right numeric operand from the left numeric operand.

5.1.2.2.3. Multiplication Operator

The Multiplication Operator (or 'mul') multiples the left and right numeric operands together.

5.1.2.2.4. Division Operator

The Division Operator (or 'div') divides the left numeric operand by the right numeric operand.

5.1.2.2.5. Modulo Operator

The Modulo Operator (or 'mod') evaluates to the remainder when the left integral operand is divided by the right integral operand.

5.1.2.2.6. Arithmetic Operator Examples

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

```
https://services.odata.org/OData/OData.svc/Products?$filter=Price add 2.45M eq '5.00M' (Requests all products with a Price of 2.5
5M).

https://services.odata.org/OData/OData.svc/Products?$filter=Price sub 0.55M eq '2.00M' (Requests all products with a Price of 2.5
5M).

https://services.odata.org/OData/OData.svc/Products?$filter=Price mul 2.0M eq '5.10M' (Requests all products with a Price of 2.55
M).

https://services.odata.org/OData/OData.svc/Products?$filter=Price div 2.55M eq '1M' (Requests all products with a Price of 2.55
M).

https://services.odata.org/OData/OData.svc/Products?$filter=Rating mod 5 eq 0 (Requests all products with a Rating exactly divisa
ble by 5).
```

5.1.2.3. Parenthesis Operator

he Parenthesis Operator (or '( )') controls the evaluation order of an expression, so that Parenthesis Operator evaluates to the expression grouped inside the parenthesis. For example:

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

Requests all products, because 9 mod 3 is 0.

5.1.2.4. Canonical Functions

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

The syntax rules for all canonical functions are defined in Appendix A.

5.1.2.4.1. substringof

The substringof canonical function has this signature:

```
Edm.Boolean substringof(Edm.String, Edm.String)
```

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

For example:

```
https://services.odata.org/Northwind/Northwind.svc/Customers?$filter=substringof('Alfreds', CompanyName) eq true
```

Returns all Customers with a CompanyName that contains ‘Alfreds’.

5.1.2.4.2. endswith

The endswith canonical function has this signature:

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

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

For example:

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

Returns all Customers with a CompanyName that end with ‘Futterkiste’.

5.1.2.4.3. startswith

The startswith canonical function has this signature:

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

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

For example:

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

Returns all Customers with a CompanyName that starts with ‘Alfr’

5.1.2.4.4. length

The length canonical function has this signature:

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

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

For example:

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

Returns all Customers with a CompanyName that is 19 characters long.

5.1.2.4.5. indexof

The length canonical function has this signature:

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

If implemented the indexof canonical function MUST return 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:

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

Returns all Customers with a CompanyName containing ‘lfreds’ starting at the second character.

5.1.2.4.6. replace

The replace canonical function has this signature:

```
Edm.String replace(Edm.String, Edm.String, Edm.String)
```

If implemented the replace canonical function MUST return the first parameter value, with all occurances of the second parameter value replaced by the third parameter value. The replaceMethodCallExpr syntax rule defines how the replace function is invoked.

For example:

```
https://services.odata.org/Northwind/Northwind.svc/Customers?$filter=replace(CompanyName, ' ', '') eq 'AlfredsFutterkiste'
```

Returns all Customers with a CompanyName that equals 'AlfredsFutterkiste' once ' ' has been replaced by ''.

5.1.2.4.7. substring

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

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

If implemented the two argument substring canonical function MUST return 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 three argument substring canonical function MUST return 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:

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

Returns all customers with a CompanyName of 'lfreds Futterkiste' once the first character has been removed.

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

Returns all customers with a CompanyName that has 'lf' as the second and third characters respectively.

5.1.2.4.8. tolower

The tolower canonical function has this signature:

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

If implemented the tolower canonical function MUST return 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:

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

Returns all Customers with a CompanyName that equals ‘alfreds futterkiste’ once any uppercase characters have been converted to lowercase.

5.1.2.4.9. toupper

The toupper canonical function has this signature:

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

If implemented the toupper canonical function MUST return the input parameter string value with all lowercase characters converted to uppercase according to unicode rules. The toUpperMethodCallExpr syntax rule defines how the tolower function is invoked.

For example:

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

Returns all Customers with a CompanyName that equals ‘ALFREDS FUTTERKISTE’ once any lowercase characters have been converted to uppercase.

5.1.2.4.10. trim

The trim canonical function has this signature:

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

If implemented the trim canonical function MUST return 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:

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

Returns all customers with a CompanyName without leading or trailing whitespace characters.

5.1.2.4.11. concat

The concat canonical function has this signature:

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

If implemented the concat canonical function MUST return a string that concatinates both input parameter string values together. The concatMethodCallExpr syntax rule defines how the concat function is invoked.

For example:

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

Returns all customers with from the City of Berlin and the Country called Germany.

5.1.2.4.12. year

The year canonical function has the following signatures:

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

If implemented the year canonical function MUST return the year component of the DateTime or DateTimeOffset parameter value. The yearMethodCallExpr syntax rule defines how the year function is invoked.

For example:

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

Returns all Employees who were born in 1971.

5.1.2.4.13. years

The years canonical function has the following signature:

```
Edm.Int32 years(Edm.Time)
```

If implemented the years the years canonical function MUST return the year component of the Time parameter value.

5.1.2.4.14. month

The month canonical function has the following signatures:

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

If implemented the month canonical function MUST return the month component of the DateTime or DateTimeOffset parameter value. The monthMethodCallExpr syntax rule defines how the month function is invoked.

For example:

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

Returns all Employees who were born in May.

5.1.2.4.15. day

The day canonical function has the following signatures:

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

If implemented the day canonical function MUST return the day component DateTime or DateTimeOffset parameter value. The dayMethodCallExpr syntax rule defines how the day function is invoked.

For example:

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

Returns all Employees who were born on the 8th day of a month.

5.1.2.4.16. days

The days canonical function has the following signature:

```
Edm.Int32 days(Edm.Time)
```

If implemented the days canonical function MUST return the days component of the Time parameter value.

5.1.2.4.17. hour

The day canonical function has the following signatures:

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

If implemented the hour canonical function MUST return the hour component of the DateTime or DateTimeOffset parameter value. The hourMethodCallExpr syntax rule defines how the hour function is invoked.

For example:

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

Returns all Employees who were born in the 4th hour of a day.

5.1.2.4.18. hours

The hours canonical function has the following signature:

```
Edm.Int32 hours(Edm.Time)
```

If implemented the hours canonical function MUST return the hours component of the Time parameter value.

5.1.2.4.19. minute

The minute canonical function has the following signatures:

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

If implemented the minute canonical function MUST return the minute component of the DateTime or DateTimeOffset parameter value. The minuteMethodCallExpr syntax rule defines how the minute function is invoked.

For example:

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

Returns all Employees who were born in the 40th minute of any hour on any day.

5.1.2.4.20. minutes

The minutes canonical function has the following signature:

```
Edm.Int32 minutes(Edm.Time)
```

If implemented the minutes canonical function MUST return the minutes component of the Time parameter value.

5.1.2.4.21. second

The second canonical function has the following signatures:

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

If implemented the second canonical function MUST return the second component of the DateTime or DateTimeOffset parameter value. The secondMethodCallExpr syntax rule defines how the second function is invoked.

For example:

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

Returns all Employees who were born in the 40th second of any minute of any hour on any day.

5.1.2.4.22. seconds

The seconds canonical function has the following signature:

```
Edm.Int32 seconds(Edm.Time)
```

If implemented the seconds canonical function MUST return the seconds component of the Time parameter value.

5.1.2.4.23. round

The round canonical function has the following signatures

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

If implemented the round canonical function MUST return round the input numeric parameter value to the nearest numeric value with no decimal component. The roundMethodCallExpr syntax rule defines how the round function is invoked.

For example:

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

Returns all Orders that have a Freight cost that rounds to 32.

5.1.2.4.24. floor

The floor canonical function has the following signatures

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

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

For example:

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

Returns all Orders that have a Freight cost that rounds down to 32.

5.1.2.4.25. ceiling

The ceiling canonical function has the following signatures

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

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

For example:

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

Returns all Orders that have a Freight cost that rounds up to 32.

5.1.2.4.26. isof

The isof canonical function has the following signatures

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

If implemented the single parameter isof canonical function MUST return true if, and only if, the current instance is assignable to the type specified. If implemented 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.

The isofMethodCallExpr syntax rule defines how the isof function is invoked.

For example:

```
https://services.odata.org/Northwind/Northwind.svc/Orders?$filter=isof(NorthwindModel.BigOrder)
```

Returns only orders that are also BigOrders.

```
https://services.odata.org/Northwind/Northwind.svc/Orders?$filter=isof(Customer, NorthwindModel.MVPCustomer)
```

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

5.1.2.4.27. cast

The cast canonical function has the following signatures:

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

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

5.1.2.5. Operator Precedence

OData Servers MUST use the following operator precedence for supported operators when evaluating $filter and $orderby expressions:

```
GROUP                   OPERATOR            DESCRIPTION

Logical Operators       eq                  Equal
                        ne                  Not Equal
                        gt                  Greater Than
                        ge                  Greater Than or Equal
                        lt                  Less Than
                        le                  Less Than or Equal
                        and                 Logical And
                        or                  Logical Or
                        not                 Logical Negation

Arithmetic Operators    add                 Addition
                        sub                 Subtraction
                        mul                 Multiplication
                        div                 Division
                        mod                 Modulo

Grouping Operators      ( )                 Precedence grouping
```

5.1.3. Expand System Query Option

The $expand system query option allows clients to request related resources when a resource that satifies a particular request is retrieved.

What follows is a snippet from [OData:ABNF][OData ABNF], that applies to the Expand System Query Option:

```
expand                        =   "$expand=" expandClause

expandClause                  =   expandItem *("," expandItem)

expandItem                    =   [ qualifiedEntityTypeName "/" ] navigationPropertyName
                                  *([ "/" qualifiedEntityTypeName ] "/" navigationPropertyName)
```

Each expandItem MUST be evaluated relative to the entity type of the request, which is the entity type of the resource(s) identified by the resource path part of the URL.

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.

The leftmost navigationPropertyName segment MUST identify a navigation property defined on the entity type of the request or an entity type derived from the entity type of the request. Subsequent navigationPropertyName segments MUST identify navigation properties defined on the entity type returned by the previous navigation property or the entity type introduced in the previous cast.

Redundant expandClause rules on the same data service URI MAY be considered valid, but MUST NOT alter the meaning of the URI.

5.1.4. Select System Query Option

The $select system query option allows clients to requests a limited set of information 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 $expand query option, to first increase the scope of the resource graph returned ($expand) and then selectively prune that resource graph ($select).

What follows is a snippet [OData ABNF][OData%20ABNF.html] that applies to the Select System Query Option:

```
select                          =   "$select=" selectClause
selectClause                    =   selectItem *( COMMA selectItem )
selectItem                      =   star /
                                    [ qualifiedEntityTypeName "/" ]
                                    (
                                        propertyName /
                                        qualifiedActionName /
                                        qualifiedFunctionName /
                                        allOperationsInContainer /
                                        ( navigationProperty [ "/" selectItem ] )
                                    )
```

The selectClause MUST be interpreted relative to the entity type or complex type of the resources identified by the resource path section of the URI, for example:

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

In this URI the “Rating,ReleaseDate” selectClause MUST be interpreted relative to the Product entity type which is the entity type of the resources identified by this https://services.odata.org/OData/OData.svc/Products URI.

Each selectItem in the selectClause indicates that the response MUST include the properties, open properties, related properties, actions and functions identified by that selectClause.

The simplest selectClause explicitly requests properties defined on the entity type of the resources identified by the resource path section of the URI, for example this URI requests just the Rating and ReleaseDate for the matching Products:

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

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

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

If a selectClause consists of a single selectItem that is a star (i.e. *), then all properties and navigation properties on the matching resources MUST be returned.

If a navigation property appears as the last segment of a selectItem and does not appear in an $expand query option, the entity or collection of entities identified by the navigation property MUST be represented as deferred content.

Each selectItem is a path, while often simply a propertyName or star, the path MAY include a cast to a derived type using a qualifiedEntityTypeName segment or a navigation to a related entity via navigation property segment followed by a nested selectItem. For example the following URI requests, the Spokesperson property of any Products that are of the derived type idenfitied by the qualifiedEntityType 'Namespace.BestSellingProduct', and the AccountRepresentative property of any related Supplier that is of a the derived type 'Namespace.PreferredSupplier':

```
http://service.odata.org/OData/OData.svc/Products?$select=Namespace.BestSellingProduct/Spokesperson,Supplier/Namespace.PreferredS
upplier/AccountRepresentative
```

If a navigation property appears as the last segment of a selectItem and the same navigation property is specified as a segment of a path in an $expand query option, then all the properties of the expanded entity identified by the selectItem MUST be in the response. In addition, all the properties of the entities identified by segments in the $expand path after the segment that matched the selectedItem MUST also be included in the response.

In order to select any nested properties of navigation properties the client MUST also include an expandClause for that navigation property. For example the following URI expands the Category navigation property so the Name of the Category can be selected.

```
https://services.odata.org/OData/OData.svc/Products?$select=Category/Name&$expand=Category
```

If a property, open property, navigation property or operation is not requested as a selectItem (explicitly or via a star), it SHOULD NOT be included in the response.

A star SHOULD NOT reintroduce actions or functions. Thus if any selectClause is specified, actions and functions SHOULD be omitted unless explicitly requested using a qualifiedActionName, a qualifiedFunctionName or the allOperationsInContainer clause.

Actions and Functions information can be explicitly requested with a selectItem containing either a qualifiedActionName or a qualifiedFunctionName or can be implicitly requested using a selectItem contain the allOperationsInContainer clause.

For example this URI requests the ID property, the 'ActionName' action defined in 'Container' and all actions and functions defined in the 'Container2' for each product, if those actions and functions can be bound to that product:

```
http://service.odata.org/OData/OData.svc/Products?$select=Container.ActionName,Container2.*
```

If an action is requested as a selectedItem, either explicitly by using a qualifiedActionName clause or implicitly by using an allOperationsInContainer clause, then for each entity identified by the last path segment in the request URI 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 selectedItem, either explicitly by using an qualifiedFunctionName clause or implicitly by using an allOperationsInContainer 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 URI, 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 selectClause, then the total set of property, open property, navigation property, actions and functions to be returned is equal to the union of the set of those identified by each selectItem.

Redundant selectClause rules on the same URI MAY be considered valid, but MUST NOT alter the meaning of the URI.

5.1.5. OrderBy System Query Option

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

The semantics of $orderby are covered in the OData:Core (../odata-version-3-0-core-protocol) document.

The OData:ABNF (../abnf) orderby syntax rule defines the formal grammar of the $orderby query option.

5.1.6. Top and Skip System Query Options

The $top system query option allows clients a required number of resources, used in conjunction $skip query option which allows a client to ask the service to begin sending resource after skipping a required number of resource, a client can request a particular page of matching resources.

The semantics of $top and $skip are covered in the OData:Core (../odata-version-3-0-core-protocol) document.

The OData:ABNF (../abnf) top and skip syntax rules define the formal grammar of the $top and $skip query options respectively.

5.1.7. Inlinecount System Query Option

The $inlinecount system query option allows clients to request a count of the number of matching resources inline with the resources in the response. Typically this is most useful when a service implements server-side paging, as it allows clients to retrieve the number of matching resources even if the service decides to only response with a single page of matching resources.

The semantics of $inlinecount is covered in the OData:Core (../odata-version-3-0-core-protocol) document.

The OData:ABNF (../abnf) inlinecount syntax rule define the formal grammar of the $inlinecount query option.

5.1.8. Format System Query Option

The $format system query option if supported allows clients to request a response in a particular format. Generally requesting a particular format is done using 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:Core (../odata-version-3-0-core-protocol) document.

The OData:ABNF (../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 Appendix A: ABNF for OData URL Conventions.

Custom query options MUST NOT begin with a “$” character because the character is reserved for system query options. A custom query option MAY begin with the “@” character, however this doing can result in custom query options that collide with function Parameters values specified using Parameter Aliases.

For example this URL addresses provide a ‘secURLtytoken’ via a custom query option: http://service.odata.org/OData/OData.svc/Products?$orderby=Name&secURLtytoken=0412312321

5.3. URL Equivalence

When determining if two URLs are equivalent, each URL SHOULD be normalized using the rules specified in RFC3987 (http://www.ietf.org/rfc/rfc3987.txt) and RFC3986 (http://) and then compared for equality using the equivalence rules specified in HTTP/1.1 (http://www.ietf.org/rfc/rfc2616.txt), Section 3.2.3.