RESO Web API v1.0.2

RESO Web API v1.0.2

Chapters

Section 1 - Proposal

1.1 Purpose

The RESO transport workgroup has been tasked with recommending a new industry-wide standard for real-time access to real estate data {directly from Web and Mobile applications}. The goal of this new standard is to provide a more open approach to data access using widely-adopted technology standards in use across industries, including the real estate industry. Specifically, the approach focuses on the use of the REST (REpresentational State Transfer) architectural approach documented by Roy Thomas Fielding and adopted by tens of thousands of developers worldwide.

The goal driving the move toward a RESTful standard for the real estate industry is to encourage and promote access to real estate information directly from Web, mobile, social and other HTTP-based applications. Using a RESTful transport will enable web applications to directly interact with RESO enabled data services. (*note: more information on RESTful can be found here*)

This workgroup sought to find an approach that does not deviate from either the solid foundations already employed from past RESO accomplishments or the existing technology standards that set out to solve similar problems for other industries.

The goals of this group were to:

1. Honor existing data service capabilities from RETS 1[1].x
2. Adopt existing standard technologies in use across industries
3. Leverage existing production-ready software toolkits

As such the group proposes the use of an existing standard that was designed specifically for data transport. The standard, Open Data Protocol or “OData” (http://www.odata.org/) serves as a set of fundamental building blocks for what the group is proposing.

The group chose OData for the following reasons:

- Well-established and robustly documented existing standard.
- Significant community adoption including “Open Government Data Initiative.”
- Well-defined functionality supports most significant RESO use cases.
- Existing open source technology implementations to support community adoption.
- Extensibility to handle specific use cases as needed.

As a standards body, we will follow the OData standard and will extend, where needed, to fulfill our industry’s needs. We will not, however, deviate from the RESTful principles, standard capabilities or query syntax that is inherent to the OData standard.

OData Overview[2]

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

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

The OData Protocol provides a uniform way to describe both the data and the data model. This improves semantic interoperability between systems and allows an ecosystem to emerge.

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

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

Further details pertaining to OData may be found at the below link: OData Version 4.0

[1] RETS 1x is a legacy protocol produced by RESO and still in use today

[2] Source: © Copyright OASIS Open 2013.

1.2 Scope

The initial scope of this standard is to support read only searching of data resources that have been defined by the Data Dictionary Workgroup and other RESO data providers.

Explicitly in scope in this initial release will be:

1. Metadata Representation
2. Read Access / Standard Search
3. Geospatial Search
4. Hypermedia Representation

Explicitly out of scope in this initial release will be:

1. Create, Update, Delete resource content
2. A Data Replication Framework
3. Requesting Binary Media Resources
4. Updating Binary Media Resources
5. Saved Searches and Resources

Explicitly out of scope for the transport specification will be:

1. Authentication and Authorization
    a. Please See the "RETS Web API Security" document.
2. The underlying Data Dictionary and Resource definitions
    a. Please see the latest "Data Dictionary" files for details.

1.3 Approach

The RESO OData Transport standardizes access to Real Estate data over the Internet using a Representational State Transfer (REST) style interface. Compatible RESO OData Transport client and server applications MUST be implemented according to the OData V4 standard specification. All further references to OData in this document refer to the OData V4 standard. Compatible server and client applications MUST send or receive data in JSON or ATOM/XML format. In keeping with OData both the client and server applications will use the standard HTTP methods GET and POST to perform the operations outlined by this document.

A compatible server takes action based on the HTTP method called by a compatible client. The following HTTP methods must be honored as follows.

- GET - gets the requested item or collection data in JSON or ATOM/XML format.
- POST - used in conjunction with X-HTTP-Method-Override header.

For POST Usage: While this is a non-standard approach, HTTP request header is the "de facto" standard for instructing a server to override the method requested with the value supplied in the header (if supported).The approach is being taken to fully leverage the existing capabilities within OData for our industry's needs.

Where possible, we will leverage existing syntax that may be augmented. Where this is not possible, new extensions will be created and may be proposed back to the OData standards group for inclusion in future releases.

In all cases, where an extension is made, a reference implementation will also be created and shared with the community.

The initial focus will be on HTTP GET for search.

The service output MUST support one of the following:

1. ATOM (XML)
2. JSON

The response format is defined by use of Content Negotiation (*http://www.w3.org/Protocols/rfc2616/rfc2616-sec12.html*) and the Accept Header may be used to define the desired data output. If Accept: */* is used the default response format is expected to be JSON. Additional formats may be supported.

Section 2 - Specification

This specification outlines the current, minimum set of functionality required by RESO as a subset of the OData V4 specification.

There is currently no reference implementation that we can go to for examples. For that reason, we highly recommend anyone not familiar with OData to use the reference implementation provided by the OData community. Their resource list can be found here: OData Community Service Endpoint.

Additionally, you can view the metadata of this reference implementation here: OData Community Reference Metadata.

The reference implementation is very useful to try out features and functionality of the OData specification even though it is not Real Estate specific.

There are also some more advanced services which can be found here: OData Services

Finally, there are some excellent examples using the TripPinService using the PostMan application which can be found here: OData PostMan Collection

2.1 Terminology

Table 1 - Terminology

Term	Definition
REST	Representational State Transfer. For more information see: http://en.wikipedia.org/wiki/Representational_state_transfer
Resource	In a RESTful API a resource is an object with a type, associated data, relationships to other resources, and a set of methods that may operate on it.
RESO Data Dictionary	A uniform set of field names and data type conventions that set a baseline across the real estate industry for how real estate data will be defined. See http://www.reso.org/data-dictionary.
Standard Resource	A data source or collection of data that is represented using the definitions found in the RESO Data Dictionary.
Custom Resource	A data source or collection of data that is represented using the something other than the RESO Data Dictionary. This may also be localized data such as language localization.
Metadata	Descriptive information about a data set, object or resource that helps a recipient understand how the data is formatted.
Payload	For purposes of the RESO community the term “payload” is synonymous with the OData term “resource.” A resource refers to the object(s) you wish to retrieve in response from the server.
Schema	A way of logically defining, grouping, organizing and structuring information so it may be understood by different systems.
MUST	This word or the adjective "required" means that the item is an absolute requirement of the specification. A feature that the specification states MUST be implemented is required in an implementation in order to be considered compliant. If the data is available in the system AND the data is presented for search then it MUST be implemented in the manner described in the specification.

SHOULD	This word or the adjective "recommended" means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course. A feature that the specification states SHOULD be implemented is treated for compliance purposes as a feature that may be implemented.
MAY	This word or the adjective "optional" means that this item is truly optional. A feature that the specification states MAY be implemented need not be implemented in order to be considered compliant. However, if it is implemented, the feature MUST be implemented in accordance with the specification.
Out of Scope	This statement means that the specific topic has not been addressed in the current specification but may be addressed in future versions.
N/A	This term means “not applicable” to the scope of this standard and will not be addressed by this standard specification.

2.2 HTTP Protocol

A compatible server implementation MUST use either HTTP or HTTPS as the protocol declared by the server URL. The version MUST be HTTP 1.1 or above. Even though OData may be written using HTTP 1.0, there are many limitations in the HTTP 1.0 specification that we want to avoid. Therefore, we are limiting compatible implementations to HTTP Version 1.1 or above.

2.2.1 Version Header

2.2.2 X-HTTP-Method-Override Header

2.2.1 Version Header

```
OData-Version: [Version]

[Version] = MAJOR.MINOR
```

The version header is used by the server to communicate the currently supported version of the specification.

- If Client Requests No version: Server MUST return the current supported version
- If Client Requests the Current version: Server MUST return the current version
- If Client Requests an Older version that the server still supports: Server MUST return requested version
- If Client Requests an Older version than the server supports: Server MUST return HTTP 400 Bad Request
- If Client Requests a Newer version than the server supports: Server MUST return HTTP 400 Bad Request

Please see 2.5 Response Message Bodies for details on expected responses.

2.2.2 X-HTTP-Method-Override Header

The X-HTTP-Method-Override allows clients making an OData request with long filter/select combinations to use the POST method and set the X-HTTP-Method-Override to GET to overcome limitations in firewalls or some web server implementations on the length of the request of the GET Method. Servers SHOULD accept the X-HTTP-Method-Override to fix this limitation.

Server vendors may still reject the request if the override method does not correspond to an appropriate Method for the resource. For example, an override of DELETE on a GET Method can be rejected. A second example of a PUT on a POST may be accepted

2.3 URL Formatting

The RESO OData Transport defines a few standardized URL formatting requirements for ease of use and application interoperability. These requirements are designed to permit standards-compliant applications and servers to interoperate in a pluggable manner requiring minimal configuration. All service URL's must match OData V4 Part 2 Section 2 URL Components in addition to the additional recommendations mentioned in this section.

2.3.1 Hostname

2.3.2 URI Stem

2.3.3 Data Systems Endpoint

2.3.4 Metadata Endpoint

2.3.5 Resource Endpoint

2.3.1 Hostname

The hostname of the URL is arbitrary and no naming convention in required. For the purposes of this standard the following example protocol and hostname will be used for clarity.

`http://odata.reso.org`

2.3.2 URI Stem

The RESO OData Transport recommends the following URI stem naming convention to simplify client application interoperability. The URI Stem is the system endpoint of the OData server as implemented by a service provider.

1. Service = http://odata.reso.org/RESO/OData/
2. Resource = http://odata.reso.org/RESO/OData/Resource
3. Resource Entity By ID = http://odata.reso.org/RESO/OData/Resource('ID')

The /RESO section denotes that a RESO standardized interface is provided.

The /OData section denotes a RESO OData Transport compliant interface is provided.

It is expected that if a client accesses the Service endpoint directly, that a response listing all the resources available is presented as per the OData specification.

An example of this would be:

```
<service xmlns="http://www.w3.org/2007/app"
xmlns:atom="http://www.w3.org/2005/Atom"
xmlns:m="http://docs.oasis-open.org/odata/ns/metadata"
xml:base="http://services.odata.org/V4/OData/OData.svc/"
m:context="http://services.odata.org/V4/OData/OData.svc/$metadata">
<workspace>
<atom:title type="text">Default</atom:title>
<collection href="Properties">
<atom:title type="text">Properties</atom:title>
</collection>
<collection href="Offices">
<atom:title type="text">Offices</atom:title>
</collection>
<collection href="Agents">
<atom:title type="text">Agents</atom:title>
</collection>
</workspace>
</service>
```

2.3.3 Data Systems Endpoint

There will be a top level URI to expose the “Data Systems” endpoint. The Data Systems end point will allow a user to inspect the Data Systems available on the service including the following details:

1. Specification Version - The version of the Transport Specification supported.
2. Data System Endpoint - The URI identifying the location of the service for that data system.
3. Available Resources - The list of available Standard or Custom Resources available in the data system.
4. Localizations of Resources - The list of available “localized” or “custom” resources that may not conform to the RESO Data Dictionary.

`http://odata.reso.org/RESO/OData/DataSystem`

This methodology permits a server to expose multiple systems as deemed appropriate. This may be used to describe a catalog of Data Systems content which a client may use.

Pleas see the following section for more information: DataSystem Resource

2.3.4 Metadata Endpoint

The $metadata endpoint is inherently defined within the DataSystems resource. To get the metadata for a given DataSystem, a client is expected to use the ServiceURi/$metadata endpoint for the DataSystem that is being accessed.

It is very important to note that a service provider may be supporting multiple systems which may each have a different set of metadata because they may be providing data for different Data Dictionary versions or different RESO Api versions.

This being the case, it is expected that clients will first execute a call to a service provider's DataSystems resource to get the list of DataSystems that are defined by the service provider. This will provide the client all the information on how to get the appropriate metadata for each DataSystem and how to access the data for each Resource provided by that DataSystem.

Possible examples of a metadata request for a specific DataSystem would be:

```
http://odata.reso.org/RESO/OData/$metadata
```

```
http://odata.reso.org/RESO/OData/SYS1/$metadata
```

```
http://odata.reso.org/RESO/OData/SYS2/$metadata
```

2.3.5 Resource Endpoint

The Resource endpoint is explicitly defined within the DataSystems resource as the ServiceURI that is output for each Resource defined within each DataSystem. Resource endpoints may be within the same DataSystem or may refer to another DataSystem that is accessible.

Clients must not attempt to execute requests against a service provider without first querying the DataSystems resource to get the information required to access each DataSystem.

Possible examples of a Resource endpoint would be:

```
http://odata.reso.org/RESO/OData/[Resource]
```

```
http://odata.reso.org/RESO/OData/SYS1/[Resource]
```

```
http://odata.reso.org/RESO/OData/SYS2/[Resource]
```

2.4 Search

Section 11 of the OData V4 specification provides full details about OData service requests and query support.

2.4.1 Search by Unique ID

Accessing a single item in a provided resource must adhere to the OData standard taking the following form:

```
http://odata.reso.org/RESO/OData/[Resource]('[ID]')
```

The [ID] must contain content conforming to the resource key as described by the resource metadata. The [ID] section is the unique ID of the requested item.

Note: You may request multiple records using the **$filter** parameter to perform a search.

2.4.2 Query Support

You can use OData queries to filter the items you get back. See System Query Option $filter for further details.

A client may retrieve a list of objects that match supplied search criteria. This is done using OData query parameters. The RESO OData Transport explicitly supports the following parameters.

- $select – MUST support
  Selects desired resource elements to be returned.
- $filter – MAY support
  Filters returned items according to filter criteria.
- $top – MUST support
  Designates the maximum number of matching items returned.
- $skip – MUST support
  Designates the number of matching items to omit before returning any items. When using $skip, it is expected that the first query sent to the server starts with a $skip=0 in order to allow servers wishing to implement consistent pagination an indication that they should prepare to receive multiple requests with differing $skip values and matching $filter.
- $orderby – MAY support
  Designates the field used to order items returned.
- $expand – MAY support
  Allows expansion of details from a related resource. Expand may only be used where the EDMX document for a Resource implements a reference.

NOTE: A server may return HTTP 413 - Request Entity Too Large if the $filter or $orderby is too complex or large for the server to process.

NOTE: Field names are case sensitive when used in the $select, $filter, and $orderby parameters. Therefore you MUST respect case sensitivity defined in the resource metadata.

2.4.3 Data Types

Although OData provides a vast array of search functionality, this version of the specification only requires a compatible server to support the following subset of Primitive Types as specified in the OData V4 Part 3 Section 4.4 Primitive Types. Microsoft has also provided a set of examples and usage for all these data types which can be found here: Open Data Protocol by Example

Type	Meaning	RESO Specific Examples
Edm.Boolean	Binary-valued logic	Waterfront, Pets Allowed
Edm.Byte	Unsigned 8-bit integer	Beds
Edm.Date	Date without a time-zone offset	List Date, Sale Date
Edm.DateTimeOffset	Date and time with a time-zone offset, no leap seconds	Open House Start
Edm.Decimal	Numeric values with fixed precision and scale	Commission
Edm.Double	IEEE 754 binary64 floating-point number (15-17 decimal digits)	Latitude, Longitude
Edm.Int16	Signed 16-bit integer	Price
Edm.Int32	Signed 32-bit integer	Price
Edm.Int64	Signed 64-bit integer	Price
Edm.SByte	Signed 8-bit integer	Level
Edm.String	Sequence of UTF-8 characters	Remarks, Area Names
Edm.TimeOfDay	Clock time 00:00-23:59:59.999999999999	
Edm.Geography	Abstract base type for all Geography types	

Edm.GeographyPoint	A point in a round-earth coordinate system	
Edm.GeographyLineString	Line string in a round-earth coordinate system	
Edm.GeographyPolygon	Polygon in a round-earth coordinate system	
Edm.GeographyMultiPoint	Collection of points in a round-earth coordinate system	
Edm.GeographyMultiLineString	Collection of line strings in a round-earth coordinate system	
Edm.GeographyMultiPolygon	Collection of polygons in a round-earth coordinate system	
Edm.EnumType	An enumeration that can represent lists of data or a single data element of a specific enumeration	Amenities, ListingStatus

2.4.4 The $filter Option

Although OData provides a vast array of filter functionality, this version of the specification only requires a compatible server to support the following subset of the Built-In Filter Operations from the OData V4 Part 1 Section 11.2.5.1.1 Built In Filter Operations section.

Operator	Description	General Example
Comparison Operators		
eq	Equal	Address/City eq 'Redmond'
ne	Not equal	Address/City ne 'London'
gt	Greater than	Price gt 20
ge	Greater than or equal	Price ge 10
lt	Less than	Price lt 20
le	Less than or equal	Price le 100
has	Has flags	Style has Sales.Color'Yellow'
Logical Operators		
and	Logical and	Price le 200 and Price gt 3.5
or	Logical or	Price le 3.5 or Price gt 200
not	Logical negation	not endswith(Description,'milk')
Grouping Operators		
( )	Precedence grouping	(Price sub 5) gt 10

Similarly, this version of the specification only requires a compatible server to support the following subset of the Built-In Query Fuctions from the OData V4 Part 1 Section 11.2.5.1.2 Built-in Query Functions section:

Function	Example
String Functions	
contains	contains(CompanyName,'freds')
endswith	endswith(CompanyName,'Futterkiste')
startswith	startswith(CompanyName,'Alfr')
tolower	tolower(CompanyName) eq 'alfreds futterkiste'
toupper	toupper(CompanyName) eq 'ALFREDS FUTTERKISTE'
Date Functions	
year	year(BirthDate) eq 0

month	month(BirthDate) eq 12
day	day(StartTime) eq 8
hour	hour(StartTime) eq 1
minute	minute(StartTime) eq 0
second	second(StartTime) eq 0
fractionalseconds	second(StartTime) eq 0
date	date(StartTime) ne date(EndTime)
time	time(StartTime) le StartOfDay
now	StartTime ge now()
Geo Functions	
geo.distance	geo.distance(CurrentPosition,TargetPosition)
geo.intersects	geo.intersects(Position,TargetArea)

2.4.5 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.

Further details on Lambda Operators can be found in the OData V4 Part 2 Section 5.1.1.5 Lambda Operators.

2.4.6 Literals

It is expected that compliant implementations of the RESO Web API adhere to the OData V4 Part 2 Section 5.1.1.6 Literals section of the OData specification with the following exceptions:

1. Support for $it
2. Support for $root

2.4.7 Geospatial Search Implementation Details

Geographic search MUST be supported using the following OData functions.

- geo.distance - Search for resources nearby
- geo.intersects - Search for resources within an area (intersection of point and area)

The geo.distance function takes two GeographyPoint objects as arguments. It is expected that the servers will allow a literal field name of type GeographyPoint to be passed in for the first GeographyPoint and that the second geography point will be the center for a radius search where the results of the geo.distance function can be compared to a specific distance. For example, a filter to ask the system to return data for all properties that are within 10 miles of a given point the filter could be geo.distance(Location, geography'SRID=4326;Point(142.1 64.1') <= 10. Note that SRID 4326 works well for all of North America.

The geo.intersects function takes a GeographyPoint and a GeographyPolygon as arguments. It is expected that the servers will allow a literal field name of type GeographyPoint to be passed in for the GeographyPoint argument so that the server can return a set of data where the specified field in the resource is within the specified GeographyPolygon. This allows geospatial queries to function on a given resource and also allows a resource to have multiple GeographyPoint data points and queries to specify a specific point to perform the operation on. For example, a filter to as the system to return data for all properties that are within a polygon using the Location field could be: geo.intersects(Location, [Any GeographyPolygon]).

Here is a good blog that describes Geospatial Properties.

2.4.8 Annotations

The metadata returned from a server may have Annotations as specified in the OData Common Schema Definition Language.

Note that Annotations are a useful way to provide more context to data and are also used in the RESO implementation to build out Multi-Valued Lookups that have more than 64 items in them.

A very useful function of Annotations is that they can be used to get around limitations in Edm.EnumType that require only certain characters in

the Name field. The following example illustrates how we use Annotations to provide meaningful names for enumerations:

Define the StandardName Annotation

```
 <edmx:Edmx Version="4.0">
 <edmx:Reference
Uri="http://standards.reso.org/transport/odata/v0.1/RESO.OData.Metadata.xm
l">
  <edmx:Include Alias="Core" Namespace="Org.OData.Core.V1"/>
 </edmx:Reference>
 <edmx:DataServices>
  <Schema Namespace="RESO.OData.Metadata">
   <Annotation Term="Core.Description">
    <String>Terms for extending OData Metadata to accommodate RESO specific
requirements</String>
   </Annotation>
   <Term Name="StandardName" Type="Edm.String" AppliesTo="EntityType
Property EnumType Member">
   <Annotation Term="Core.Description">
    <String>The standard name of the entity, property, enumeration, or
enumeration value</String>
   </Annotation>
   </Term>
  </Schema>
 </edmx:DataServices>
</edmx:Edmx>
```

The above annotation declares the term 'StandardName' so it can be used to describe an EntityType, Property, EnumType or enumeration Member as needed.

Implementing 'Nice' names for EnumType Members

```
 <?xml version="1.0" encoding="utf-8"?>
<edmx:Edmx Version="4.0"
xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx">
 <edmx:DataServices>
  <Schema Namespace="My.Property.Data"
xmlns="http://docs.oasis-open.org/odata/ns/edm">
   <EntityType Name="MyProperties">
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Property"
/>

    <Key>
     <PropertyRef Name="ID" />
    </Key>
    <Property Name="ID" Type="Edm.Int32" Nullable="false" />
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Property
ID" />
    <Property Name="Type" Type="My.Property.Data.Type" Nullable="false" />
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Property
Type" />
    <Property Name="Features" Type="My.Property.Data.FeaturesType"
Nullable="true" />
```

```
     <Annotation Term="RESO.OData.Metadata.StandardName"
String="LotFeatures" />
   </EntityType>

   <EnumType Name="RecordType">
    <Annotation Term="RESO.OData.Metadata.StandardName"
String="PropertyType" />
    <Member Name="RES" Value="1" />
     <Annotation Term="RESO.OData.Metadata.StandardName"
String="Residential Property" />
    <Member Name="MUL" Value="2" />
     <Annotation Term="RESO.OData.Metadata.StandardName"
String="Multi-Family Property" />
    <Member Name="COM" Value="3" />
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Commercial
Property" />
   </EnumType>

   <EnumType Name="FeaturesType" IsFlags='true'>
    <Annotation Term="RESO.OData.Metadata.StandardName"
String="LotFeatures" />
    <Member Name="Adjacent_to_Wash" Value="1" />
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Adjacent
to Wash" />
    <Member Name="Alley" Value="2"/>
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Alley" />
    <Member Name="Auto_Timer_H2O_Back" Value="4" />
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Auto Timer
H2O Back" />

[interim values omitted for brevity]

    <Member Name="Pool" Value="4611686018427387904" />
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Pool" />
   </EnumType>
```

```
  </Schema>
 </edmx:DataServices>
</edmx:Edmx>
```

Notice that in the above example, we can have nice StandardNames in many different places such as the field names of a resource and the descriptions of single-valued or multi-valued Enumerations.

2.4.9 Single-Valued Lookups

A Single Valued Lookup is a field that can have one and only one selection from a list of values. The Web API Implements Single Valued Lookup fields using the OData data type of Edm.EnumType with an underlying type of Edm.Int32. The values returned for data of this Type MUST be those that are identified in the RESO Data Dictionary as per the supporting EDMX metadata for the specific field in the resource. No additional selections for the field may be provided by a system. If additional selections are required, these MUST only be output in a Localized resource (see the DataSystem Resource section for more details about Localized Resources).

An example of a Single-Valued Lookup might be:

```
<EnumType Name="StandardStatus" UnderlyingType="Edm.Int32">
  <Member Name="Active" Value="1" />
  <Member Name="Active Under Contract"  Value="2" />
  <Member Name="Pending" Value="3" />
  <Member Name="Hold" Value="4" />
  <Member Name="Withdrawn" Value="5" />
  <Member Name="Closed" Value="6" />
  <Member Name="Expired" Value="7" />
  <Member Name="Cancelled" Value="8" />
  <Member Name="Delete" Value="9" />
  <Member Name="Incomplete" Value="10" />
  <Member Name="Coming Soon" Value="11" />
</EnumType>
```

Any server implementing a field that is an Edm.EnumType must strictly follow the definition of the enumeration adhering to both the Name and Value provided by the standard EDMX document containing the specified type.

2.4.10 Multi-Valued Lookups

A Multi-Valued Lookup is a field that can have one or more items selected from a list of values. Multi-Valued Lookups MUST adhere to all the limitations enforced by the Single Valued Lookups with the addition of the IsFlags="true" attribute being specified which indicates that a bit-map field implementation is being used to manage the lookup. The UnderlyingType will be Edm.Int32 for a Multi-Valued Lookup with 32 or fewer choices and Edm.Int64 for more than 32 and 64 or fewer choices. The special case of greater than 64 choices is described below.

An example of a Multi-Valued lookup might be:

```
<EnumType Name="AssociationFeeIncludes" UnderlyingType="Edm.Int32"
IsFlags="true">
  <Member Name="Cable_TV"   Value="1" />
  <Member Name="Earthquake_Insurance"  Value="2" />
  <Member Name="Electricity" Value="4" />
  <Member Name="Gas" Value="8" />
  <Member Name="Insurance" Value="16" />
  <Member Name="Maintenance_Exterior" Value="32" />
  <Member Name="Maintenance_Grounds" Value="64" />
  <Member Name="Pest_Control" Value="128" />
  <Member Name="Security" Value="256" />
  <Member Name="Sewer" Value="512" />
  <Member Name="Snow Removal" Value="1024" />
  <Member Name="Trash" Value="2048" />
  <Member Name="Utilities" Value="4096" />
  <Member Name="Water" Value="8192" />
</EnumType>
```

If a Multi-Valued Lookup has more than 64 choices, two or more Multi-Valued Lookups will be defined and Annotations will be used to relate the Multi-Valued Lookup definitions so that they can be combined for display and selection. This is simply due to the limitation of the OData implementation of EnumType when IsFlags is set to true.

At the time this document was being written, there are only two enumerations in the Data Dictionary 4 specification that require more than 64 items. To see how we would use Annotations to solve this problem, please examine the following example:

```
 <?xml version="1.0" encoding="utf-8"?>
<edmx:Edmx Version="4.0"
xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx">
 <edmx:DataServices>
  <Schema Namespace="My.Property.Data"
xmlns="http://docs.oasis-open.org/odata/ns/edm">
   <EntityType Name="MyProperties">
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Property"
/>
    <Key>
     <PropertyRef Name="ID" />
    </Key>
    <Property Name="ID" Type="Edm.Int32" Nullable="false" />
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Property
ID" />
    <Property Name="PropertyType" Type="My.Property.Data.PropertyType"
Nullable="false" />
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Property
Type" />
    <Property Name="LotFeatures_1"
Type="My.Property.Data.LotFeatures_1_Type" Nullable="true" />
     <Annotation Term="RESO.OData.Metadata.StandardName"
String="LotFeatures" />
    <Property Name="LotFeatures_2"
Type="My.Property.Data.LotFeatures_2_Type" Nullable="true" />
     <Annotation Term="RESO.OData.Metadata.StandardName"
String="LotFeatures" />
```

```
   </EntityType>

   <EnumType Name="LotFeatures_1_Type">
    <Annotation Term="RESO.OData.Metadata.StandardName"
String="LotFeatures" />
    <Member Name="Adjacent_to_Wash" Value="1">
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Adjacent
to Wash" />
    </Member>
    <Member Name="Alley" Value="2">
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Alley" />
    </Member>
    <Member Name="Auto_Timer_H2O_Back" Value="4">
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Auto Timer
H2O Back" />
    </Member>
[additional values omitted for brevity]
    <Member Name="Pool" Value="4611686018427387904">
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Pool" />
    </Member>
    <Member Name="RearYard" Value="9223372036854775808">
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Rear Yard"
/>
    </Member>
    <Member Name="RearyrdLot" Value="18446744073709551616">
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Rear Yard
Lot" />
    </Member>
   </EnumType>

   <EnumType Name="LotFeatures_2_Type">
    <Annotation Term="RESO.OData.Metadata.StandardName" String="Lot
Features" />
    <Member Name="RoughGraded" Value="1">
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Rough
Graded" />
    </Member>
    <Member Name="RuralLot" Value="2">
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Rural Lot"
/>
    </Member>
    <Member Name="Security_Entrance" Value="4">
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Security
Entrance" />
    </Member>
[additional values omitted for brevity]
    <Member Name="WterViewLot" Value="16777216">
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Water View
Lot" />
    </Member>
    <Member Name="Yrd_Wtring_Sys_Back" Value="33554432">
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Yard
Watering System Back" />
```

```
    </Member>
    <Member Name="Yrd_Wtring_Sys_Front" Value="67108864">
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Yard
Watering System Front" />
    </Member>
   </EnumType>
```

```
    </Schema>
   </edmx:DataServices>
  </edmx:Edmx>
```

The goal here is to use the StandardName of LotFeatures to tell a client that two separate Enumerations with different field names are related to each other. A client presenting the RESO Data Dictionary value LotFeatures enumeration choices may choose to present the Data Dictionary field LotFeatures as a single field instead of two fields, LotFeatures_1 and LotFeatures_2 based on the fact that they have the same StandardName. In this case, the client will combine the LotFeatures_1 and LotFeatures_2 enumerations into a single display field LotFeatures in the client application. A client MUST still submit queries to the server using the individual fields LotFeatures_1 and LotFeatures_2 to provide the user-selected choices to the server. A client presenting search results for LotFeatures may choose to combine the LotFeatures_1 and LotFeatures_2 data values into a single display field LotFeatures.

The example above also illustrates how to use Annotations to give a nice, human readable, name to Enumeration choices. This is necessary as no spaces or special characters may be used in values for an enumeration.

This workaround was selected by the Transport Workgroup as the simplest way to maintain OData compatibility while allowing lists that have more than 64 choices. The Transport Workgroup will be following up with the OData organization to see if this limitation can be removed so that we can get around this workaround.

2.5 Response Message Bodies

2.5.2 HTTP Response Codes

2.5.3 Error Message Bodies

2.5.2 HTTP Response Codes

A compatible server implementation MUST return a valid HTTP status code for each request indicating the status of the request when ATOM-XML is requested. If the response was not successful the server MAY include an error message in the body of the HTTP response. There is a defined response body for JSON but there is no explicit requirement in the OData standard.

See Section 2.5.3 for response details.

Table 3 - HTTP Response Codes

Code	Short Description	Detail
200	OK	Returned by GET method when retrieving a record or records. If no records are found an empty result set is returned.
202	Accepted	Returned by GET method to indicate tht the server received the request but that it may take time to fulfill a response.
400	Bad Request	Returned by GET method calls when the data fails validation and more detail on the error may be found in the body of the response.
403	Forbidden	Returned when the selected Authentication mechanism is not successful.
404	Not Found	Returned when a GET cannot find a resource or collection.
413	Request Entity Too Large	Returned at the discretion of the server. Used to indicate when the server cannot handle the complexity of the specific request.
415	Unsupported Media	Returned when a media format requested is not supported by the system.
429	Too Many Requests	Returned at the discretion of the server. Used to indicate that the user / licensee has met or exceeded their allowed usage (transactions per second, per day, per month, etc.
500	Internal Server Error	Returned when an unexpected error is encountered and more detail may be provided in the response body.
501	Not Implemented	Returned when the requested method is not available.

2.5.3 Error Message Bodies

When the client makes a request which cannot be satisfied or produces an error condition, a compliant server MUST follow the OData error handling guidelines specified by the ATOM format. Full details of this mechanism may be found in the ATOM format specification at the following URL.

http://www.odata.org/documentation/odata-v3-documentation/atom-format/#15_Errors_as_XML

The following example includes a client request and a compliant server error response for reference.

Example Client Request:

```
http://odata.reso.org/RESO/OData/Members.svc/Members?$orderby=MemberID&$top=5&$skip=5
```

Example Server Response:

```
Sample 1 - Error Message Body from Server
```

```
<error xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
  <code>NOSKIP</code>
  <message xml:lang="en-US">Bad Request - Resource does not support $skip parameter</message>
</error>
```

Servers are encouraged to put as much detail in the message body of errors to give users the best chance of understanding and dealing with th errors.

2.6 Standard Resources

In general, it is expected that the RESO Web API be able to output data as per the Standard Data Dictionary.

This section describes any standard resources that are to be implemented to aid the end-user in using the system.

The RESO OData Transport only defines one data resource for use with the transport standard, the DataSystem resource.

In addition to the mandatory DataSystem resource, at least ONE of the following resources from the RESO Data Dictionary MUST be supported by any compliant server:

1. Property - a “Property” resource based on the 1.1 data dictionary and MAY support other versions of the dictionary.
2. Member - a “Member” resource based on the 1.1 data dictionary and MAY support other versions of the dictionary.
3. Office - an “Office” resource based on the 1.1 data dictionary and MAY support other versions of the dictionary.
4. Media - a “Media” resource based on the 1.1 data dictionary and MAY support other versions of the dictionary.

Servers MAY support more than one version of a response and may also define additional resources to support specific use cases. For example, a server could provide a “Mobile” resource that returns a condensed list of fields to reduce the size of a response. Servers may also support “custom” or “localized” resources that may not follow the RESO Data Dictionary Standard as long as they are identified as such in the DataSystem resource.

2.6.1 DataSystem Resource

This resource MUST be provided by all implementations. The goal of this resource is to provide a list of URI's for one or more systems or sets of MLS Data that may be available from a single RESO OData provider. If a system is providing data for a single system, then it is expected that there be a single DataSystem output when the DataSystem endpoint is navigated to. The XML Schema that defines the DataSystem resource can be found here: **Appendix 4 - DataSystem XML Schema**

To get the list of DataSytems provided by a server, the URI Stem of the server followed by /DataSystem is used.

The DataSystem resource defines the following top level fields:

Field Name	Description
ID	The unique key of the data system. This can be used in a query for the specific DataSystem being requiested. ie: http://odata.reso.org/RESO/OData/DataSystems(5) would return information about the DataSystem with the primary key of 5.
Name	A unique identifier that describes the Name of the DataSystem.
ServiceURI	This is to be considered the URI Stem of the DataSystem. For example, if a system were to support data from two separate systems named SYS1 and SYS2, they would be expected to have two separate DataSystem records which might have the following two different ServiceURI's: http://odata.reso.org/RESO/OData/SYS1 and http://odata.reso.org/RESO/OData/SYS2. This essentially defines two separate systems which can have different $metadata by appending the ServiceURI with the $metadata keyword. If a system is designed to only support a single DataSystem, then the ServiceURI should be the same as the URI Stem of the server.

DateTimeStamp	The last modification date of the $metadata within the DataSystem.
TransportVersion	This is expected to be the API Version of the RESO Web API that has been implemented and must be in the form: VersionMajor.VersionMinor.VersionRelease of the RESO Web API.
DataDictionaryVersion	The Data Dictionary Version of the Data System. It is expected that all non-localized resources provided by this DataSystem adhere to this version of the Data dictionary. This version must be in the form: VersionMajor.VersoinMinor of the RESO Data Disctionary version that has been implemented by the Data System.
Resources	The list of Resources with the data fields as defined in the Data System Resources Collection table. All Resources that are custom and specific to the DataSystem are to be identified as a Localizations and not as Resources since Resources may only be those as defined in the RESO Data Dictionary.

The Data System Resources Collection defines the following fields:

Field Name	Description
Name	The unique name of the Resource within the DataSystem.
ResourcePath	This is the ResourcePath that is to be appended after the ServiceURI of the DataSystem when getting data for that resource. This is generally expected to be the same as the Name of the Resource, but is allowed to be different for flexibility purposes. For example, if the ServiceURI is http://odata.reso.org/RESO/OData/SYS1, the following URI would be used to get data for the 'Property' resource: http://odata.reso.org/OData/SYS1/Property.
Description	A description of the Resource expected to be a human readable explanation of what data is provided by the resource.
DateTimeStamp	The last modification date of the $metadata within the Resource.
TimeZoneOffset	The TimeZone Offset provided in standard TimeZone notation of GMT[+|-]X, where X is the number of hours in the offset.
Localizations	The list of related Resources that provide localized data that is related to this Resource. The data fields for this collection are defined in the Localizations Collection table. A Localization can be either a list of complementary fields that further define the base Resource or they may be a complete set of fields that include all the data in a localized format. If the list is a complimentary set of data, it is expected that an appropriate OData reference be defined in the metadata of the Localization.

The Localizatoins Collection defines the following fields:

Field Name	Descrpition
Name	The unique name of the Localization within the DataSystem.
ResourcePath	This is the ResourcePath that is to be appended after the ServiceURI of the DataSystem when getting data for this localized resource. This is generally expected to be the same as the Name of the Localization, but is allowed to be different for flexibility purposes. For example, if the ServiceURI is http://odata.reso.org/RESO/OData/SYS1, the following URI would be used to get data for the 'Residential' localization: http://odata.reso.org/OData/SYS1/Residential.
Description.	A description of the Localization expected to be a human readable explanation of what data is provided by the resource.
DateTimeStamp	The last modification date of the $metadata within the Localization.

The DataSystem resource should not be confused with the $metadata OData command that can be used at the ServiceURI of the server. The $metadata is standard OData metadata that defines all resources supported at the ServiceURI along with the fields within the resource and all relationships between resources in the system. It is important to note that a system supporting multiple different Data Systems may only provide limited metadata if the $metadata qualifier is used at the URI Stem of the server. Typically, on a server that supports more than one system, the $metadata at the URI Stem of the server will only return the metadata describing the DataSystem resource. On a server supporting only a single system, the $metadata qualifier at the URI Stem of a server may return all the metadata for all data within the system. For portability, it is strongly recommended that clients first execute a request for the Data Systems (ie: http://odata.reso.org/RESO/OData/DataSystems) to get the full list of systems being supported and then use the $metadata qualifier on the ServiceURI that is returned for the DataSystem that is being worked with. This ensures functionality with both multi-system implementations and single-system implementations.

2.6.2 Data Dictionary Resources

Following the list of RESO Data Dictionary standard resources with links to their EDMX definitions that must be adhered to for all of these standard resources.

Resource	EDMX Definition
Office	
Agent	
Property	
Media	
OpenHouse	
Rooms	
Units	

Section 3 - Security

Authentication and authorization is not covered in this document. It is expected that implementations will follow the standard recommendation from the RESO Web API Security v1.0.2.

Section 1 - Intro to OpenID Connect

Section 2 - FAQ

Section 4 - Authors

Section 5 - Revision List

Section 6 - Appendices

Section 4 - Authors

Author	Company
Scott Petronis	Onboard Informatics
Matthew McGuire	CoreLogic
Sergio Del Rio	Templates for Business, Inc.
Fred Larsen	UtahRealEstate.com
James McDaniel	UtahRealEstate.com
Robert Gottesman	RESO

RESO Transport Workgroup

Section 5 - References

Table 4 - Document References

Description	Link
REST	http://en.wikipedia.org/wiki/Representational_state_transfer
Open Data Protocol or “OData”	http://www.odata.org/
OData V4	http://www.odata.org/documentation
OData V4 - Part 1 - Protocol	http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part1-protocol.html
OData V4 - Part 2 - URL Conventions	http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part2-url-conventions.html
OData V4 - Part 3 - Common Schema Definition Language	http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part3-csdl.html
Geospatial Support in OData	http://www.odata.org/blog/geospatial-data-support-in-odata/
RFC2616 HTTP Protocol 1.1	http://www.ietf.org/rfc/rfc2616.txt

Section 6 - List of Tables & Figures

Section 6 - List of Tables & Figures

Tables

Samples

Section 7 - Revision List

Revision Comment	Revision Date	Revision By
Addition of Section 2.6	2015-03-19	Sergio Del Rio
Fixed Discrepancies in Appendix 3 - DataSystem XML Schema	2015-03-19	Sergio Del Rio
Updated to OData V4 throughout document	2015-04-01	Sergio Del Rio
Section 2.2 - Cleaned up References to ATOM and JSON	2015-04-01	Sergio Del Rio
Section 2.2.1 - Removed RESO prefix to OData-Version header	2015-04-01	Sergio Del Rio
Section 2.3.3 - Now reference the new Section 2.6	2015-04-01	Sergio Del Rio
Section 2.5.2 - Addition of 202, 413 and 429 response codes	2015-04-01	Sergio Del Rio
Section 2.4.2 - Added $expand and changed $filter to MAY	2015-04-01	Sergio Del Rio
Section 2.4.3 - Cleaned up the Enumeration wording	2015-04-03	Sergio Del Rio
Section 2.2 - Cleaned up more, left the specifics to the OData V4 specification instead of trying to detail them here.	2015-04-03	Sergio Del Rio
Section 2.4 - Major Changes - Revised Entire Section	2015-04-03	Sergio Del Rio
Section 2.5.3 - Added comment encouraging servers to add good error message text.	2015-04-03	Sergio Del Rio
Section 2.5.1 - Deleted - Moved to 2.6 Standard Resources section	2015-04-03	Sergio Del Rio
Section 3 - Updated link and included page to API Security v1.0.1	2015-04-06	Cal Heldenbrand
Section 2.3.3, 2.3.4 and 2.3.5 - Edited to better explain the DataSystem resource and how it impacts clients.	2015-05-11	Sergio Del Rio
Added SRID to Geo section and added extra link to Data Types section.	2015-05-26	Sergio Del Rio
Revised bad links as OData moved their main documentation link yet again. Revised several sections to include a few more details and examples. Also added a few more places to get some useful OData examples from.	2015-08-02	Sergio Del Rio
Section 2.4.9 - Clarified wording.	2015-08-11	Sergio Del Rio
Section 2.4.10 - Clarified wording.	2015-08-11	Sergio Del Rio
Section 1.3 - Changed wording of output formats.	2015-08-12	Sergio Del Rio
Section 2.4.10 - Fixed Typo	2016-04-28	Sergio Del Rio

Section 8 - Appendices

Appendix 1 - Use Cases

MUST = Must support this functionality**.**
SHOULD = Should support this functionality based on proposed approach.
MAY = May support this functionality but no proposed approach. Roadmap item.
N/A = Not available in first release and no proposed approach. May be a roadmap item.

Table 5 - Use Cases

UC ID#	Category	Use Cases / Functionality	Example	Required
1	001 - Listing Search	Listing search by geography (name)	Akron, Ohio	MUST
2	001 - Listing Search	Listing search by point + radius	(long, lat) 20 unit = miles	MUST
3	001 - Listing Search	Listing search by boundary	polygon, multi-polygon	MUST
4	001 - Listing Search	Listing search by address + radius	123 Main St. Akron Ohio 20 unit=miles	MUST
5	001 - Listing Search	Listing search by specific address	123 Main St. Akron Ohio	MUST
6	001 - Listing Search	Listing search by specific street	Main St. Akron Ohio	MUST
7	001 - Listing Search	Listing search by map bounds	Upper left, lower right, what falls within	MUST
8	003 - Other Search	Retrieve listing details by ID	Give me the details of this specific listing	MUST
9	005 - Group Search	Get count of listings by geography (name)	How many listings are there in Chicago, IL (ZIP Code, County, Neighborhood, etc.)	MUST
10	010 - Authentication	Authentication	Give me access to the API	N/A
11	010 - Authentication	Authorization	What data do I have access to?	N/A
12	010 - Authentication	Authorization	What capabilities do I have access to?	N/A
13	010 - Authentication	Authorization	Give me access to this specific data.	N/A
14	020 - Media	Get specific media for this specific listing (agent) or record	Give me the main photo, thumbnail, etc. (urls)	MUST
15	020 - Media	Get all media for this group of listings (agent) or record	Give me the main photo urls for all these listings	MUST
16	030 - Metadata	What data dictionaries does the server support	Can I request 1.0, 1.1, 1.2, etc., names? What other resources are supported (non-RESO)?	MUST

17	030 - Metadata	MLS Rules	What are the specific rules for this MLS?	N/A
18	030 - Metadata	MLS Rules, retrieve specific info	Give me the disclaimer, logo, copyright, etc.	N/A
19	030 - Metadata	Record rules	What can I do with this specific record?	N/A
20	040 - Agent / Office	Agent or office search by geography (name)	Akron, Ohio (See lines 1-9)	MUST
21	040 - Agent / Office	Agent or office search by point + radius	(long, lat) 20 unit = miles (See lines 1-9)	SHOULD
22	040 - Agent / Office	Agent or office search by boundary	polygon, multi-polygon (See lines 1-9)	SHOULD
23	040 - Agent / Office	Agent or office search by address + radius	123 Main St. Akron Ohio 20 unit=miles (See lines 1-9)	SHOULD
24	040 - Agent / Office	Agent or office search by specific address	123 Main St. Akron Ohio (See lines 1-9)	SHOULD
25	040 - Agent / Office	Agent or office search by geography (name)	Akron, Ohio (See lines 1-9)	MUST
26	040 - Agent / Office	Agent or office search by map bounds	Upper left, lower right, what falls within (See lines 1-9)	SHOULD
27	040 - Agent / Office	Retrieve agent or office details by ID	Give me the details of this specific agent or office (See lines 1-9)	MUST
28	040 - Agent / Office	Get count of agents/offices by geography (name)	How many agents/offices are there in Chicago, IL (See lines 1-9)	MAY
29	050 - Open House	Open house search by date range and geography (name)	Akron, Ohio (See lines 1-9)	MUST
30	050 - Open House	Open house search by date range and point + radius	(long, lat) 20 unit = miles (See lines 1-9)	SHOULD
31	050 - Open House	Open house search by date range and boundary	polygon, multi-polygon (See lines 1-9)	SHOULD
32	050 - Open House	Open house search by date range and address + radius	123 Main St. Akron Ohio 20 unit=miles (See lines 1-9)	SHOULD
33	050 - Open House	Open house search by date range and specific address	123 Main St. Akron Ohio (See lines 1-9)	SHOULD
34	050 - Open House	Open house search by date range and specific street	Main St. Akron Ohio (See lines 1-9)	MAY
35	050 - Open House	Open house search by date range and map bounds	Upper left, lower right, what falls within (See lines 1-9)	SHOULD
36	050 - Open House	Retrieve open house details by date range and ID	Give me the details of this specific open house (See lines 1-9)	MUST
37	050 - Open House	Get count of open houses by date range and geography (name)	How many open houses are there in Chicago, IL (See lines 1-9)	N/A
38	060 - Statistics	Search by Statistics	Count of listings with price reductions between 5-10% in the last 30 days in specified zip codes	Out of Scope

39	080 - System	Manage Pagination	Identify the number of records per page and the specific page they would like to get back.	MUST
40	080 - System	Retrieve system capabilities, metadata	Query system to determine what types of resources, search capabilities, record limits and other constraints there may be.	MUST
41	001 - Listing Search	Simple result sortation	Allow the user to select the desired sort based on a single field (e.g. <field> ascending or descending)	MUST
42	001 - Listing Search	Advanced sort	Allow the user to apply multiple sort rules on multiple fields (e.g. sort by this, then by this)	MUST
43	001 - Listing Search	Preferential sort	Bring "my" listings to the top then sort the rest by the simple or advanced sort criteria	Out of scope
44	001 - Listing Search	Search by multiple boundaries	Bring back listings that are within any of the provided boundaries.	MUST
45	001 - Listing Search	Search in boundary intersection	Bring back listings that are within the boundary created by the intersection of two or more boundaries	MUST
46	001 - Listing Search	Saved search	"Push" content to the user based on pre-selected "search" criteria	Out of scope
47	001 - Listing Search	Alerts	Alert a "subscriber" when a listing becomes available in a specific area.	Out of scope
48	001 - Listing Search	Exclude listings with specific attributes	I don't want short sales or beach front	MUST
49	070 - Resource	Request Just IDs (Keys)	I only want to bring back the IDs of the records matching my request.	MUST
50	070 - Resource	Request Defined Resource	I want to bring back a specific resource (e.g., Full IDX, Mobile, VOW, Syndication, etc.) for the records matching my request.	MUST
51	070 - Resource	Request Specific Fields	I want to bring back only the specific fields I indicate for the records matching my request.	MUST
52	011 - Edit	Modify specific listing attributes	As an agent I want to modify specific listing attributes from my mobile device.	Out of scope
53	020 - Media	Retrieve additional documents pertaining to a listing or other record.	As an agent I want to retrieve documents such as disclosures, HOA minutes and other related documents pertaining to a listing. (This is another type of media)	MUST
54	001 - Listing Search	Keep my local database synchronized (replication) against a remote data store via an API	As an application developer I want to be able to request updates to my local data from one or more MLSs	Out of scope
55	001 - Listing Search	Aggregate data from multiple sources for local storage	As an application developer I want to be able to request updates to my local data from one or more MLSs and keep track of source details	Out of scope

Appendix 2 - Basic Query Examples

This appendix provides a set of example queries using OData V4 and the specific RESO resources discussed in this document. This is intended to highlight various common use cases, not to describe all the possible queries that may be executed. It is important that all URL's must be properly URL Encoded when they are sent to the server. We have intentionally not done this in the examples in order to make the examples more human readable.

1 - Request the list of Data Systems

2 - Select a single data system

3 - How do I look at the metadata for a specific service?

4 - How do I retrieve data using this metadata?

5 - Get a single Property

6 - Select specific field values

7 - Filter by field value

8 - Filter by multiple field values

1 - Request the list of Data Systems

To get the complete list of DataSystems provided by a server, append /DataSystem to the URI Stem of the server.

An example of this would be:

```
http://odata.reso.org/RESO/OData/DataSystem
```

Each Data System provided by the service will be listed as <entry> items. The client may select a single DataSystem using the ID of the desired DataSystem.

2 - Select a single data system

```
http://odata.reso.org/RESO/OData/DataSystem('RESO_MLS')?format=atom
```

The client selects a specific single Data System. The resulting XML for this is verbose so only the relevant parts of the response are shown here.

Sample 5 - Select a single data system

```
<entry>
<id>http://odata.reso.org/DataSystem.svc/DataSystem('RESO_MLS')</id>
<category term="RESO.OData.Transport.DataSystem"
scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
<link rel="edit" title="DataSystem" href="DataSystem('RESO_MLS')" />
<title />
<updated>2013-11-22T19:13:50Z</updated>
<author>
<name />
</author>
<content type="application/xml">
 <m:properties>
 <d:Name>RESO_MLS</d:Name>
 <d:ServiceURI>http://odata.reso.org/DataSystem.svc/</d:ServiceURI>
 <d:DateTimeStamp m:type="Edm.DateTime">2013-11-22T14:13:50.3810781-05:00</d:DateTimeStamp>
 <d:TransportVersion>0.9</d:TransportVersion>
 <d:Resources m:type="Collection(RESO.OData.Transport.Resource)">
 <d:element>
  <d:Name>Property</d:Name>
  <d:ResourcePath>Property</d:ResourcePath>
  <d:Description>RESO Standard Property Resource</d:Description>
  <d:DateTimeStamp
m:type="Edm.DateTime">2013-11-22T14:13:50.3810781-05:00</d:DateTimeStamp>
  <d:TimeZoneOffset m:type="Edm.Int32">-5</d:TimeZoneOffset>
```

http://odata.reso.org/RESO/OData/DataSystem('RESO_MLS')?$format=json

The same content is available in JSON format as well and the above example will look like the following one in JSON format.

Sample 6 - Select a single data system w/JSON

```
{
"odata.metadata": "http://odata.reso.org/DataSystem.svc/$metadata#DataSystem/@Element",
"Name": "RESO_MLS",
"ServiceURI": "http://odata.reso.org/DataSystem.svc/",
"DateTimeStamp": "2013-11-22T17:41:34.8131432-05:00",
"TransportVersion": "0.9",
"Resources": [{
"Name": "Property",
"ResourcePath": "Property",
"Description": "RESO Standard Property Resource",
"DateTimeStamp": "2013-11-22T17:41:34.8131432-05:00",
"TimeZoneOffset": -5,
...etc...
```


3 - How do I look at the metadata for a specific service?

For a server that implements a single system, the metadata can be retrieved by adding the $metadata argument to the URI Stem of the server as follows:

`http://odata.reso.org/RESO/OData/$metadata`

If the server is a single-system implementation, then this should return the metadata for all resources exposed by that system.

On the other hand, if the server supports multiple systems, all you should expect to receive is the metadata of the standard DataSystem resource. This might look something like the following:

```
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx"
Version="4.0">
<edmx:DataServices>
<Schema xmlns="http://docs.oasis-open.org/odata/ns/edm"
Namespace="ODataService">
<EntityType Name="DataSystem">
<Key>
<PropertyRef Name="ID" />
</Key>
<Property Name="ID" Type="Edm.Int32" Nullable="false" />
<Property Name="Name" Type="Edm.String" />
<Property Name="ServiceURI" Type="Edm.String" />
<Property Name="DateTimeStamp" Type="Edm.DateTimeOffset" />
<Property Name="TransportVersion" Type="Edm.Int32" />
<Property Name="DataDictionaryVersion" Type="Edm.Int32" />
<Property Name="Resources" Type="Collection(ODataService.Resource)" />
</EntityType>
<ComplexType Name="Resource">
<Property Name="ID" Type="Edm.Int32" Nullable="false" />
<Property Name="ResourceID" Type="Edm.Int32" Nullable="false" />
<Property Name="Name" Type="Edm.String" />
<Property Name="ServiceURI" Type="Edm.String" />
<Property Name="Description" Type="Edm.String" />
<Property Name="DateTimeStamp" Type="Edm.DateTimeOffset" />
<Property Name="TimeZoneOffset" Type="Edm.String" />
<Property Name="Localizations" Type="Collection(ODataService.Localization)"
/>
</ComplexType>
<ComplexType Name="Localization">
<Property Name="ResourceID" Type="Edm.Int32" Nullable="false" />
<Property Name="ClassID" Type="Edm.Int32" Nullable="false" />
<Property Name="Name" Type="Edm.String" />
<Property Name="Description" Type="Edm.String" />
<Property Name="ServiceURI" Type="Edm.String" />
<Property Name="DateTimeStamp" Type="Edm.DateTimeOffset" />
</ComplexType>
</Schema>
<Schema xmlns="http://docs.oasis-open.org/odata/ns/edm"
Namespace="Default">
<EntityContainer Name="Container">
<EntitySet Name="DataSystems" EntityType="ODataService.DataSystem" />
</EntityContainer>
</Schema>
</edmx:DataServices>
</edmx:Edmx>
```

For servers that support multiple systems, the metadata for each individual system can be obtained by using the ServiceURI of the System followed by the $metadata argument, which might look like this:

`http://odata.reso.org/RESO/OData/SYS1/$metadata`

This is now expected to output the metadata for all resources that are supported by that specific System, including all localizations.

4 - How do I retrieve data using this metadata?

Once you have the metadata for a given system, a client may begin interacting with the server to search for the desired data for any supported resource.

This is accomplished by using the ServiceURI that was used to get the metadata, followed by a path to the resource being searched for.

Specific search examples are provided following this section in the appendix.

5 - Get a single Property

```
http://odata.reso.org/RESO/OData/SYS1/Property('ListingId3')?$format=atom
```

Here is a truncated example response for the request above.

Sample 9 - Get Single Property return ATOM XML

```
<?xml version="1.0" encoding="utf-8"?>
<entry xml:base="http://odata.reso.org/Properties.svc/"
xmlns="http://www.w3.org/2005/Atom"
xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
xmlns:georss="http://www.georss.org/georss" xmlns:gml="http://www.opengis.net/gml">
  <id>http://odata.reso.org/Properties.svc/Properties('ListingId3')</id>
  <category term="CoreLogic.DataService.RESO.Property"
   scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
<link rel="edit" title="Property" href="Properties('ListingId3')" />
<title />
 <updated>2013-11-14T21:27:24Z</updated>
<author>
    <name />
</author>
<content type="application/xml">
  <m:properties>
    <d:ID>ListingId3</d:ID>
    <d:AboveGradeFinishedArea m:type="Edm.Single">3</d:AboveGradeFinishedArea>
    <d:AboveGradeFinishedAreaSource>AboveGradeFinishedAreaSource3
    </d:AboveGradeFinishedAreaSource>
    <d:AboveGradeFinishedAreaUnits>AboveGradeFinishedAreaUnits3
    </d:AboveGradeFinishedAreaUnits>
...etc...
```

The client may change this to JSON as well as follows:

```
http://odata.reso.org/RESO/OData/SYS1/Property('ListingId3')?$format=json
```

This will return the following example result again truncated for brevity.

Sample 10 - Change the response to JSON

```
{
    "odata.metadata":
"http://odata.reso.org/Properties.svc/$metadata#Properties/@Element",
    "ID": "ListingId3",
    "AboveGradeFinishedArea": 3,
    "AboveGradeFinishedAreaSpecified": false,
    "AboveGradeFinishedAreaSource": "AboveGradeFinishedAreaSource3",
    "AboveGradeFinishedAreaUnits": "AboveGradeFinishedAreaUnits3",
    "AccessibilityFeatures":
        ["AccessibilityFeatures1",
        "AccessibilityFeatures2",
        "AccessibilityFeatures3"],
    "AdditionalParcelsDescription": "AdditionalParcelsDescription3",
    "AdditionalParcelsYN": "AdditionalParcelsYN3",
    "ApprovalStatus": "ApprovalStatus3",
    "ArchitecturalStyle": "ArchitecturalStyle3",
...etc...
```

6 - Select specific field values

```
http://odata.reso.org/RESO/OData/SYS1/Member?$select=MemberLastName,MemberFirstName,MemberID
```

Note: *All names in the $select option are case-sensitive to match the names of elements provided by the resource.*

7 - Filter by field value

```
http://odata.reso.org/RESO/OData/SYS1/Member?$filter=(MemberLastName eq 'Smith')
```

Note: All names in the $filter option are case-sensitive to match the names of elements provided by the resource.

8 - Filter by multiple field values

```
http://odata.reso.org/RESO/OData/SYS1/Member?$filter=(MemberFirstName eq 'Joe' and MemberLastName eq
'Smith').
```

Note: *Query strings MUST be URL encoded where appropriate by a compliant client.*

9 - Get the first five Members

```
http://odata.reso.org/RESO/OData/SYS1/Member?$orderby=MemberID$top=5
```

10 - Get the second five Members

```
http://odata.reso.org/RESO/OData/SYS1/Member?$orderby=MemberID&$top=5&$skip=5
```

Note: *The implementation of $top and $orderby is defined by the server and may restrict what values may be used in either option. A compliant client SHOULD use the $orderby query to sustain consistency between requests, however a compliant server is not required to guarantee consistent results between requests.*

11 - Get the top ten Residential properties within 1 mile of a specific point ordered by distance

```
http://odata.reso.org/RESO/OData/Property?$filter=/PropertyType/Name eq "Residential" and
geo.distance(Location,POINT(-127.89 45.23)) lt 1&$orderby=geo.distance(Location, POINT(-127.89
45.23))&$top=10
```

12 - Get all the properties with a price range of $250k to $500k within a specific area drawn on map (polygon)

```
http://odata.reso.org/RESO/OData/Property?$filter=ListPrice gt 250000 and ListPrice lt 500000 and
geo.intersects(Location,POLYGON((-127.01 45.50,-127.00 45.49,-127.01 45.49,-127.00 45.50)))
```

13 - Get all the properties with a price range of $250k to $500k within the map on the screen (polygon)

```
http://odata.reso.org/RESO/OData/Property?$filter=ListPrice gt 250000 and ListPrice lt 500000 and
geo.intersects(Location,POLYGON((-127.02 45.08,-127.02 45.38,-127.32 45.38,-127.32 45.08,-127.02 45.08)))
```

14 - Get all properties with price range of $250k to $500k within a complex drawn area on map (multi-polygon)

```
http://odata.reso.org/RESO/OData/Property?$filter=ListPrice gt 250000 and ListPrice lt 500000 and
geo.intersects(Location,MULTIPOLYGON(((-127.02 45.08,-127.023 45.38,-127.32 45.38,-127.32 45.08,-127.02
45.08)),((-127.12 45.18,-127.12 45.28,-127.22 45.28,-127.22 45.18,-127.12 45.28))))
```

15 - Get all the Residential properties within a half mile of a specific road (linestring)

```
http://odata.reso.org/RESO/OData/Property?$filter=PropertyType/Name eq "Residential" and
geo.distance(Location, LINESTRING (-118.62 34.22, -118.61 34.22, -118.61 34.21, -118.62 34.2, -118.62
34.22)) lt 0.5
```

16 - Request only IDs

```
http://odata.reso.org/RESO/OData/Property?$filter=ID
```

17 - Get all the properties with a listing price less than $300K

```
http://odata.reso.org/RESO/OData/Property?$filter=ListPrice lt 300000
```

18 - Get all the properties with a listing price greater than $300K

```
http://odata.reso.org/RESO/OData/Property?$filter=ListPrice gt 300000
```

19 - Get all the properties with a listing price of $300K

```
http://odata.reso.org/RESO/OData/Property?$filter=ListPrice eq 300000
```

20 - Query using boolean to find all properties that are short sales

```
http://odata.reso.org/RESO/OData/Property?$filter=ShortSale eq true
```

21 - Combine multiple criteria in a search

```
http://odata.reso.org/RESO/OData/Property?$filter=ListPrice gt 250000 and ListPrice lt 500000
```

22 - Get records back in a certain order

```
http://odata.reso.org/RESO/OData/Property?$filter=ListPrice lt 300000&$orderby=ListPrice desc
```

23 - Get a count of records

```
http://odata.reso.org/RESO/OData/Property?$filter=ListPrice lt 300000&$inlinecount=allpages
```

24 - Get all members whose first name starts with ‘Joh’

```
http://odata.reso.org/RESO/OData/Member?$filter=startswith(MemberFirstName, 'Joh')
```

25 - Get all members whose last name ends with ‘ith’

```
http://odata.reso.org/RESO/OData/Member?$filter=endswith(MemberLastName, 'ith')
```

26 - Get all members whose last name contains the string ‘ohns’

```
http://odata.reso.org/RESO/OData/Member?$filter=indexof(MemberLastName, 'ohns')
```

27 - Get all members whose first name is ‘James’ or ‘Adam’ and who are active

```
http://odata.reso.org/RESO/OData/Member?$filter=(MemberStatus eq 'Active' and (MemberFirstName eq 'James'
or MemberFirstName eq 'Adam'))
```

28 - Get all properties that were listed in the year 2013

```
http://odata.reso.org/RESO/OData/Property?$filter=year(ListDate) eq 2013
```

29 - Get all properties that were listed in May of 2013

```
http://odata.reso.org/RESO/OData/Property?$filter=year(ListDate) eq 2013 and month(ListDate) eq 5
```

30 - Get records by enumeration (lookup) values

An enumeration or lookup is a data field that contains one or more string values. The following examples detail how to search for records containing specific string values.

The lookup is AccessibilityFeatures and I want to search for records that have HandicapAccess as one of the items:

```
http://odata.reso.org/RESO/OData/Property?$filter=AccessibilityFeatures/any(a: a eq 'HandicapAccess')
```

The ‘a: a’ represents the content (or predicate) that is being tested in the statement where the letter ‘a’ follows the ‘:’ character. Since the AccessibilityFeatures property is a string collection this means ‘a’ is a string representing one of the values. Please note that the any() operation is declared following the AccessibilityFeatures/ because the property represents a collection of strings.

31 - Get records by values in a contained collection

A given record may contain a property that itself contains a collection of items. This is useful for situations such as Rooms and Units. The following example uses Rooms and the details apply to any collections contained within a given resource.

The Rooms property is a collection of Room elements and these are complex types which contain underlying properties. In this case the any function again uses the predicate to designate what is being inspected by the statement. Therefore if the predicate is a complex type the properties of that type become accessible to the statement. For example criteria for a Room with the Room/Type equal to ‘Living’ will look like the following:

```
http://odata.reso.org/RESO/OData/Property?$filter=Rooms/any(a: RoomType eq 'Living')
```

The all() operation uses the exact same syntax with the additional requirement that ALL items in the collection must match the criteria or it will return false.

Appendix 3 - Advanced Query Examples

Appendix 4 - DataSystem XML Schema

Figure 1 - DataSystem XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.reso.org/RESO/DataSystem"
    xmlns:tns="http://www.reso.org/RESO/DataSystem"
    elementFormDefault="unqualified">

<complexType name="DataSystem">
        <sequence>
            <element name="Name" type="tns:Name"></element>
            <element name="ID" type="tns:ID"></element>
         <element name="ServiceURI" type="anyURI"></element>
            <element name="DateTimeStamp" type="tns:DateTimeStamp"></element>
            <element name="DataDictionaryVersion"
type="tns:DataDictionaryVersion"></element>
            <element name="TransportVersion" type="tns:TransportVersion"></element>
      <element name="Resources" type="tns:Resources"></element>
        </sequence>
    </complexType>
    <complexType name="Resources">
        <sequence>
            <element name="Resource" type="tns:Resource"
            minOccurs="1" maxOccurs="unbounded"></element>
        </sequence>
    </complexType>
    <complexType name="Resource">
        <sequence>
            <element name="Name" type="tns:Name"></element>
            <element name="ResourcePath" type="tns:ResourcePath"></element>
            <element name="Description" type="tns:Description"></element>
            <element name="DateTimeStamp" type="tns:DateTimeStamp"></element>
            <element name="TimeZoneOffset" type="tns:TimeZoneOffset"></element>
            <element name="Localizations" type="tns:Localizations"></element>
        </sequence>
    </complexType>
    <complexType name="Localizations">
        <sequence>
            <element name="Localization" type="tns:Localization"
            minOccurs="0" maxOccurs="unbounded"></element>
        </sequence>
```

```
    </complexType>
    <complexType name="Localization">
        <sequence>
            <element name="Name" type="tns:Name"></element>
            <element name="ResourcePath" type="ths:ResourcePath"></element>
            <element name="Description" type="tns:Description"></element>
            <element name="DateTimeStamp" type="tns:DateTimeStamp"></element>
        </sequence>
    </complexType>
    <simpleType name="ID">
        <restriction base="string"></restriction>
    </simpleType>
    <simpleType name="Name">
        <restriction base="string"></restriction>
    </simpleType>
    <simpleType name="ResourcePath">
        <restriction base="string"></restriction>
    </simpleType>
    <simpleType name="ServiceURI">
        <restriction base="anyURI"></restriction>
    </simpleType>
    <simpleType name="Description">
        <restriction base="string"></restriction>
    </simpleType>
    <simpleType name="DateTimeStamp">
        <restriction base="dateTime"></restriction>
    </simpleType>
    <simpleType name="TimeZoneOffset">
        <restriction base="int"></restriction>
    </simpleType>
    <simpleType name="TransportVersion">
        <restriction base="string"></restriction>
    </simpleType>
    <simpleType name="DataDictionaryVersion">
        <restriction base="string"></restriction>
    </simpleType>
</schema>
```

The following is a sample OData XML EDMX instance of the schema for reference.

Figure 2 - OData XML EDMX instance of the schema

```
<edmx:Edmx xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx" Version="1.0">
  <edmx:DataServices xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
m:DataServiceVersion="3.0" m:MaxDataServiceVersion="3.0">
   <Schema xmlns="http://schemas.microsoft.com/ado/2009/11/edm" Namespace="RESO.OData.Transport">
    <EntityType Name="DataSystem">
   <Key>
<PropertyRef Name="ID"/>
</Key>
<Property Name="Name" Type="Edm.String"/>
<Property Name="ServiceURI" Type="Edm.String"/>
<Property Name="DateTimeStamp" Type="Edm.DateTime" Nullable="false"/>
<Property Name="TransportVersion" Type="Edm.String"/>
<Property Name="DataDictionaryVersion" Type="Edm.String"/>
<Property Name="Resources" Type="Collection(RESO.OData.Transport.Resource)" Nullable="false"/>
<Property Name="ID" Type="Edm.String" Nullable="false"/>
</EntityType>
   <ComplexType Name="Resource">
<Property Name="Name" Type="Edm.String"/>
<Property Name="ResourcePath" Type="Edm.String"/>
<Property Name="Description" Type="Edm.String"/>
<Property Name="DateTimeStamp" Type="Edm.DateTime" Nullable="false"/>
<Property Name="TimeZoneOffset" Type="Edm.Int32" Nullable="false"/>
<Property Name="Localizations" Type="Collection(RESO.OData.Transport.Localization)" Nullable="false"/>
</ComplexType>
   <ComplexType Name="Localization">
<Property Name="Name" Type="Edm.String"/>
<Property Name="ResourcePath" Type="Edm.String"/>
<Property Name="Description" Type="Edm.String"/>
<Property Name="DateTimeStamp" Type="Edm.DateTime" Nullable="false"/>
</ComplexType>
</Schema>
</edmx:DataServices>
</edmx:Edmx>
```

The following is a sample OData XML (ATOM) encapsulated instance of the schema for reference.

Figure 3 - OData XML (ATOM) encapsulated instance of the schema for reference

```
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
osoft.com/ado/2007/08/dataservices/metadata" xmlns:georss="http://www.georss.org/georss" xmlns:gml="http:
se="http://localhost:2099/DataSystem.svc/">
 <id>http://localhost:2099/DataSystem.svc/DataSystem</id>
 <title type="text">DataSystem</title>
 <updated>2014-04-11T15:24:00Z</updated>
 <link rel="self" title="DataSystem" href="DataSystem"/>
 <entry>
  <id>http://localhost:2099/DataSystem.svc/DataSystem('RESO_MLS')</id>
  <category term="RESO.OData.Transport.DataSystem" scheme="http://schemas.microsoft.com/ado/2007/08/datas
  <link rel="edit" title="DataSystem" href="DataSystem('RESO_MLS')"/>
  <title/>
  <updated>2014-04-11T15:24:00Z</updated>
  <author>
   <name/>
  </author>
  <content type="application/xml">
   <m:properties>
    <d:Name>RESO_MLS</d:Name>
    <d:ServiceURI>http://odata.reso.org/DataSystem.svc/</d:ServiceURI>
    <d:DateTimeStamp m:type="Edm.DateTime">2014-04-11T11:24:00.6508563-04:00</d:DateTimeStamp>
    <d:TransportVersion>0.9</d:TransportVersion>
    <d:DataDictionaryVersion>1.3</d:DataDictionaryVersion>
    <d:Resources m:type="Collection(RESO.OData.Transport.Resource)">
     <d:element>
      <d:Name>Property</d:Name>
      <d:ResourcePath>Property</d:ResourcePath>
      <d:Description>RESO Standard Property Resource</d:Description>
      <d:DateTimeStamp m:type="Edm.DateTime">2014-04-11T11:24:00.6508563-04:00</d:DateTimeStamp>
      <d:TimeZoneOffset m:type="Edm.Int32">-5</d:TimeZoneOffset>
      <d:Localizations m:type="Collection(RESO.OData.Transport.Localization)">
       <d:element>
        <d:Name>Single Family</d:Name>
        <d:ResourcePath>SingleFamily</d:ResourcePath>
        <d:Description>Localized Single Family Residential Resource</d:Description>
        <d:DateTimeStamp m:type="Edm.DateTime">2014-04-11T11:24:00.6508563-04:00</d:DateTimeStamp>
       </d:element>
       <d:element>
        <d:Name>Multi Family</d:Name>
        <d:ResourcePath>MultiFamily</d:ResourcePath>
        <d:Description>Localized Multi Family Residential Resource</d:Description>
        <d:DateTimeStamp m:type="Edm.DateTime">2014-04-11T11:24:00.6508563-04:00</d:DateTimeStamp>
       </d:element>
      </d:Localizations>
     </d:element>
     <d:element>
      <d:Name>Office</d:Name>
      <d:ResourcePath>Office</d:ResourcePath>
      <d:Description>RESO Standard Office Resource</d:Description>
      <d:DateTimeStamp m:type="Edm.DateTime">2014-04-11T11:24:00.6508563-04:00</d:DateTimeStamp>
      <d:TimeZoneOffset m:type="Edm.Int32">-5</d:TimeZoneOffset>
      <d:Localizations m:type="Collection(RESO.OData.Transport.Localization)"/>
     </d:element>
     <d:element>
      <d:Name>Member</d:Name>
      <d:ResourcePath>Member</d:ResourcePath>
      <d:Description>RESO Standard Member Resource</d:Description>
      <d:DateTimeStamp m:type="Edm.DateTime">2014-04-11T11:24:00.6508563-04:00</d:DateTimeStamp>
      <d:TimeZoneOffset m:type="Edm.Int32">-5</d:TimeZoneOffset>
      <d:Localizations m:type="Collection(RESO.OData.Transport.Localization)"/>
     </d:element>
    </d:Resources>
    <d:ID>RESO_MLS</d:ID>
   </m:properties>
  </content>
 </entry>
</feed>
```

The following is a sample OData JSON encapsulated instance of the schema for reference.

Figure 4 - OData JSON encapsulated instance of the schema for reference

```
}
"odata.metadata": "http://localhost:2099/DataSystem.svc/$metadata",
"value": [{
   "Name": "RESO_MLS",
   "ServiceURI": "http://odata.reso.org/DataSystem.svc/",
   "DateTimeStamp": "2014-04-11T12:02:48.509401-04:00",
   "TransportVersion": "0.9",
   "DataDictionaryVersion": "1.3",
   "Resources": [{
      "Name": "Property",
      "ResourcePath": "Property",
      "Description": "RESO Standard Property Resource",
      "DateTimeStamp": "2014-04-11T12:02:48.509401-04:00",
      "TimeZoneOffset": -5,
      "Localizations": [{
            "Name": "Single Family",
            "ResourcePath": "SingleFamily",
            "Description": "Localized Single Family Residential
Resource",
            "DateTimeStamp": "2014-04-11T12:02:48.509401-04:00"
      },
      {
            "Name": "Multi Family",
            "ResourcePath": "MultiFamily",
            "Description": "Localized Multi Family Residential
Resource",
            "DateTimeStamp": "2014-04-11T12:02:48.509401-04:00"
      }]
   },
   {
      "Name": "Office",
      "ResourcePath": "Office",
      "Description": "RESO Standard Office Resource",
      "DateTimeStamp": "2014-04-11T12:02:48.509401-04:00",
      "TimeZoneOffset": -5,
      "Localizations": []
   },
   {
      "Name": "Member",
      "ResourcePath": "Member",
      "Description": "RESO Standard Member Resource",
      "DateTimeStamp": "2014-04-11T12:02:48.509401-04:00",
      "TimeZoneOffset": -5,
      "Localizations": []
   }],
   "ID": "RESO_MLS"
}]

}
```