RESO Web API v1.0.3

RESO Web API v1.0.3

Chapters

Section 1 - Proposal

1.1 Purpose

1.2 Scope

1.3 Approach

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 (**RE**presentational **S**tate **T**ransfer) 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. For specific HTTP references, please see the Section 5 - References section.

Since the RESO Web API requires that the RESO Web API Security v1.0.3 be used for authorization and authentication, we also require that all server implementations must implement TLS Security. For specific TLS references, please see the Section 5 - References section.

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 is 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 /resosection 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 ATOM 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://odata.reso.org/reso/odata/"
         m:context="http://odata.reso.org/reso/odata/$metadata">
<workspace><atom:title type="text">Default</atom:title>
   <collection href="Membesr">
      <atom:title type="text">Members</atom:title>
   </collection>
   <collection href="Offices">
      <atom:title type="text">Offices</atom:title>
   </collection>
   <collection href="Properties">
      <atom:title type="text">Properties</atom:title>
   </collection>
   <collection href="OpenHouses">
      <atom:title type="text">OpenHouses</atom:title>
   </collection>
   <collection href="Media">
      <atom:title type="text">Media</atom:title>
   </collection></workspace>
</service>
```

A JSON example would be:

```
{
@odata.context: "http://odata.reso.org/reso/odata/$metadata",
value: [{name: "Members", kind: "EntitySet", url: "Members"},
        {name: "Offices", kind: "EntitySet", url: "Offices"},
        {name: "Properties", kind: "EntitySet", url: "Properties"},
        {name: "OpenHouses", kind: "EntitySet", url: "OpenHouses"},
        {name: "Media", kind: "EntitySet", url: "Media"},
       ]
}
```

2.3.3 Data Systems Endpoint

There will be a top level URI to expose the “Data Systems” end point. 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/DataSystems`

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.

Please 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:

```
https://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 – MUST 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

<table>
<tr><th>Type</th><th>Meaning</th><th>RESO Specific Examples</th></tr>
<tr><td>Edm.Boolean</td><td>Binary-valued logic</td><td>Waterfront, Pets Allowed</td></tr>
<tr><td>Edm.Byte</td><td>Unsigned 8-bit integer</td><td>Beds</td></tr>
<tr><td>Edm.Date</td><td>Date without a time-zone offset</td><td>List Date, Sale Date</td></tr>
<tr><td>Edm.DateTimeOffset</td><td>Date and time with a time-zone offset, no leap seconds</td><td>Open House Start</td></tr>
<tr><td>Edm.Decimal</td><td>Numeric values with fixed precision and scale
<table><tr><th>Attribute</th><th>Meaning</th></tr><tr><td>Precision</td><td>• Is the maximum number of significant digits allowed in the property’s value
• It MUST be a positive integer
• If no value is specified the decimal property has unspecified precision
• MUST be a non-negative integer between 0 and 12 for a temporal property</td></tr><tr><td>Scale</td><td>• Is the maximum number of digits allowed to the right of the decimal point
• May be a non-negative integer or "variable"
  • Integer Value
  • The number of digits to the right of the decimal point may vary from 0 to the value of Scale
  • The number of digits to the left of the decimal point may vary from 1 to the value of (Precision - Scale). If Precision == Scale, then a single 0 must precede the decimal point
  • Scale must be <= Precision. If no value is specified, Scale defaults to 0
  • Variable
  • The number of digits to the right of the decimal point may vary from zero to the value of the Precision</td></tr></table></td><td>Commission</td></tr>
<tr><td>Edm.Double</td><td>IEEE 754 binary64 floating-point number (15-17 decimal digits)</td><td>Latitude, Longitude</td></tr>
<tr><td>Edm.Int16</td><td>Signed 16-bit integer</td><td>Price</td></tr>
<tr><td>Edm.Int32</td><td>Signed 32-bit integer</td><td>Price</td></tr>
<tr><td>Edm.Int64</td><td>Signed 64-bit integer</td><td>Price</td></tr>
<tr><td>Edm.SByte</td><td>Signed 8-bit integer</td><td>Level</td></tr>
<tr><td>Edm.String</td><td>Sequence of UTF-8 characters
<table><tr><th>Attribute</th><th>Meaning</th></tr><tr><td>MaxLength</td><td>• Is the maximum length of the string</td></tr></table></td><td>Remarks, Area Names</td></tr>
<tr><td>Edm.TimeOfDay</td><td>Clock time 00:00-23:59:59.999999999999</td><td></td></tr>
<tr><td>Edm.Geography</td><td>Abstract base type for all Geography types</td><td></td></tr>
<tr><td>Edm.GeographyPoint</td><td>A point in a round-earth coordinate system</td><td></td></tr>
</table>

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 requires that a compatible server MUST 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 requires that a compatible server MUST support the following subset of the Built-In Query Functions 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)
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>
    <Property Name="Type" Type="My.Property.Data.Type" Nullable="false">
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Property
Type" />
    </Property>
    <Property Name="Features" Type="My.Property.Data.FeaturesType"
```

```
Nullable="true">
     <Annotation Term="RESO.OData.Metadata.StandardName"
String="LotFeatures" />
    </Property>
   </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>
    <Member Name="MUL" Value="2">
     <Annotation Term="RESO.OData.Metadata.StandardName"
String="Multi-Family Property" />
    </Member>
    <Member Name="COM" Value="3">
     <Annotation Term="RESO.OData.Metadata.StandardName" String="Commercial
Property" />
    </Member>
   </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>
    <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>

[interim values omitted for brevity]

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

Some databases have an option of creating these kinds of fields as a bitmap fields and others do not. A bitmap field is a field where each bit in the numeric value of the field represents a distinct value. These implementations allow for extremely fast performance and, as such, are often the preferred method for dealing with Multi Valued lookups. Other database implementations either have an external reference table for a list of values by way of a foreign key to such a table.

In order to satisfy both use-cases, servers can implement either method and clients must be able to deal with both implementations.

2.4.10.1 Multi Valued Lookups - Bitmap Fields

A Bitmap Field Multi Valued Lookup is a field that can have one or more items selected from a list of values. Bitmap 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.

2.4.10.2 Multi Valued Lookups - Collections of Enumerations

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. A field that contains Multi-Valued lookups must make use of an Enumeration that has IsFlags=false. The UnderlyingType of the enumeration must be either Edm.Int32 or Edm.Int64 depending on the size of the values that are being returned.

A field that contains Multi-Valued Lookups based on the defined Enumeration must be defined as a Collection of Enumerations.

Unlike Bitmap Multi Valued Lookups, the values can be any valid number based on the Edm.Int32 or Edm.int64 data type and do not need to only consist of values that represent each possible in the number.

Examples:

Example Enumeration Definition

```
<EnumType Name="Association_Amenities" IsFlags="false"
UnderlyingType="Edm.Int64">
<Member Name="Banquet_Facilities" Value="50041194459" /> <Annotation
Term="RESO.OData.Metadata.MRIS.StandardName" <String>Banquet
Facilities</String> </Annotation>
<Member Name="Barbecue" Value="50041194461" />
<Member Name="Biking_Trails" Value="50041194463" /> <Annotation
Term="RESO.OData.Metadata.MRIS.StandardName" <String>Biking Trails</String>
</Annotation>
</EnumType>
```

Field Using Enumeration

```
<Property Name="AssociationAmenities"
Type="Collection(RESO.OData.Metadata.MRIS.Association_Amenities)">
<Annotation Term="RESO.OData.Metadata.MRIS.StandardName">
<String>Association Amenities</String> </Annotation>
</Property>
```

Query: Get Rows with both Biking Trails and Gated Parking

```
https://services.dev.mris.com/RESO/OData/MRIS/Property?$format=json&$filte
r=(AssociationAmenities/any(a: a eq
RESO.OData.Metadata.MRIS.Association_Amenities'Biking_Trails') and
AssociationAmenities/any(a: a eq
RESO.OData.Metadata.MRIS.Association_Amenities'Gated_Parking'))
```

Query: Get Rows with either Biking Trails and Gated Parking

```
https://services.dev.mris.com/RESO/OData/MRIS/Property?$format=json&$filte
r=(AssociationAmenities/any(a: a eq
RESO.OData.Metadata.MRIS.Association_Amenities'Biking_Trails') or
AssociationAmenities/any(a: a eq
RESO.OData.Metadata.MRIS.Association_Amenities'Gated_Parking'))
```

2.5 Response 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 that 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.

Full details of this mechanism may be found in the ATOM and JSON format specification at the following URLs:

JSON Error Response

ATOM Error Response

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 - JSON Error Message Body from Server

```
{
  "error": {
    "code": "501",
    "message": "Unsupported functionality",
    "target": "query",
    "details": [
      {
       "code": "301",
       "target": "$skip",
       "message": "Resource does not support the $skip parameter"
      }
    ],
    "innererror": {
      "trace": [...],
      "context": {...}
    }
  }
}
```

Sample 2 - ATOM Error Message Body from Server

```
<error xmlns="http://docs.oasis-open.org/odata/ns/metadata">
  <code>501</code>
  <message>Unsupported functionality</message>
  <target>query</target>
  <details>
    <detail>
      <code>301</code>
      <message>Resource does not support the $skip parameter</message>
      <target>$skip</target>
    </detail>
  </details>
</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 the 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 Specification MUST be supported by any compliant server:

1. Property - a “Property” resource based on the RESO Data Dictionary Specification.
2. Member - a “Member” resource based on the RESO Data Dictionary Specification.
3. Office - an “Office” resource based on the RESO Data Dictionary Specification.
4. Media - a “Media” resource based on the RESO Data Dictionary Specification.

Servers MAY support more than one version of the RESO Data Dictionary 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 Specification as long as they are identified as such in the DataSystem resource.

2.6.1 Data System Resource

This resource MAY 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 Data System output when the Data System end point is navigated to. The XML Schema that defines the Data System resource can be found here: **Appendix 4 - Data System XML Schema**

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

The Data System 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 Data System being requested. ie: http://odata.reso.org/reso/odata/DataSystems(5) would return information about the Data System with the primary key of 5.
Name	A unique identifier that describes the Name of the Data System.
ServiceURI	This is to be considered the URI Stem of the Data System. 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 Data System 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 Data System, then the ServiceURI should be the same as the URI Stem of the server.
DateTimeStamp	The last modification date of the $metadata within the Data System.

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 Data System adhere to this version of the Data dictionary. This version must be in the form: VersionMajor.VersionMinor 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 Data System 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 Data System.
ResourcePath	This is the ResourcePath that is to be appended after the ServiceURI of the Data System 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/reso/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	Data type is Edm.DateTimeOffset (the offset portion carries both hours and minutes). For example: 2015-12-01T00:00:00-08:00
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 Localizations Collection defines the following fields:

Field Name	Descrpition
Name	The unique name of the Localization within the Data System.
ResourcePath	This is the ResourcePath that is to be appended after the ServiceURI of the Data System 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/reso/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 Data System 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 Data System 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 System (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 Data System that is being worked with. This ensures functionality with both multi-system implementations and single-system implementations.

2.6.2 Data Dictionary Resources

The Web API is intended to facilitate data exchange within a market and across different markets. This makes data sharing more efficient and creating applications that use the data less expensive. To accomplish this, many common field terms, data types and some classes of values are expressed in the RESO Data Dictionary, a separate related standard from RESO.

The Web API is intended, but not restricted, to use resources named in the RESO Data Dictionary. These resources have specific names for

certain fields in a resource and specific names for values in certain cases for enumerations.

This section provides guidance to implementors of this standard on how to handle the resources, fields and enumerations of the RESO Data Dictionary to maximize interoperability between implementations.

The standard MAY enforce some or all of these guidance items in the certification testing. Please refer to the RESO Web API Server Testing Rules v1.0.3 for further information.

This section uses the terminology of the Odata V 4.0.1 standard. Please refer to Odata V4.0 Data Model for more information. The Odata term is written in *italics.*

Odata Term	Data Dictionary Term	Example
Entity Set	Resource	Properties
Entity		Property
declared property	field	ListingKey
Enumeration	enumeration type	Status
enumeration named constant	enumeration value	Sold

2.6.2.1 Entity Set Names

When an implementation has an *Entity Set* that is substantially similar to a resource name defined in the Data Dictionary, the implementation MUST use the Data Dictionary resource name for the corresponding *Entity Set* name. Implementations MAY have additional *Entity Sets* that are not defined in the Data Dictionary to meet the needs of the implementation.

2.6.2.2 declared property Names

Within a *Entity Set* that matches a Data Dictionary resource, when an implementation has a *declared property* name that is substantially similar to a field name defined in the Data Dictionary, the implementation MUST use the Data Dictionary field name for the *declared property* name in the XML and Json representation of the *Entity Set*. When a *declared property* does not have an equivalent Data Dictionary field, an Annotation tag SHOULD be used to indicate that the *declared property* is not part of the compliance set for the Data Dictionary. Failure to use an Annotation tag may result in a compliance failure or warning.

Client applications SHOULD inspect any *declared property* with respect to data type, and where a data type does not match the expected Data Dictionary value, the application SHOULD expect to find the correct data type in the Annotation tag. Client applications may need special handling to deal with these cases while the industry transitions historical data to new data types.

Within a Data Dictionary *Entity Set*, many implementations will have one or more *declared property* is not part of the Data Dictionary standard. In these cases, implementors and their customers are encouraged to have the *declared property* included in the Data Dictionary where appropriate. Some *declared property* will remain specific to an implementation based on locale, business rules or other considerations. *declared property* of this type MUST have an Annotation attached to indicate that this *declared property* is specific to the implementation.

Within an *Entity Set* that is not part of any Data Dictionary resource, implementors are encouraged to identify any *declared property* names that are identical to one defined for a different Data Dictionary resource and to use that *declared property* name in this *Entity Set*. The *Entity Set* SHO ULD have an Annotation attached to the *Entity Set* to indicate that this is specific to the implementation. The *declared property* SHOULD have an Annotation attached to each to indicate that this is specific to the implementation. Implementors may omit the Annotation when the *declared property* matches the Data Dictionary field. Implementors MUST NOT define a *declared property* that has a semantically different meaning that a Data Dictionary field name.

2.6.2.3 Enumeration

Within an *Entity Set*, when an implementation has an *Enumeration* that is substantially similar to an enumeration type in the Data Dictionary, the implementation MUST use the Data Dictionary name for the *Enumeration* name.

2.6.2.4 Enumeration named constant

Within an *Entity Set*, when an implementation has an *enumeration* with a *named constant* that is substantially similar to an enumeration value in the Data Dictionary, the implementation MUST use the Data Dictionary enumeration value for the *enumeration named constant* name.

2.6.2.5 Extending *Entity Set, Enumeration and Enumeration named constant*

In many implementations, specific *Entity Set*, *Enumeration* and *enumeration named constant* that are not part of the Data Dictionary will be required. As described in 2.6.2.2 for Entity Names, extensions are indicated by an Annotation on the *Entity Set*, *Enumeration* or *enumeration named constant*.

2.6.2.6 Large Enumerations

Odata in the current and previous versions has a limitation on the size of a multi-value enumeration. Refer to section 2.4.8 Annotations for further details.

2.6.2.7 Annotation Term Namespace

To simplify processing, this standard suggests that the Annotation Term represent a namespace and follow a recommended form. In the examples below, the form suggested has a root of RESO.OData.Metadata. Each implementation will append the MLSName as appropriate for the instance.

2.6.2.8 Examples

1. Data Dictionary Compliant

```
<EntityType Name="Property"> <!-- A Data Dictionary Resource Name -->
<Key>
  <PropertyRef Name="ListingKey" />
</Key>
<Property Name="ListingKey" Type="Edm.Int32" Nullable="false" /> <!-- A
Data Dictionary Entity Name -->
```

2. Data Dictionary Non-Compliant Data Type

```
<EntityType Name="Property"> <!-- A Data Dictionary Resource Name -->
<Key>
  <PropertyRef Name="ListingKey" />
</Key>
<Property Name="ListingKey" Nullable="false" Type="Edm.String"
MaxLength="255"> <!-- A Data Dictionary Entity Name -->
<Annotation Term="RESO.OData.Metadata.StandardName" String="ListingKey" />
<!-- Non compliant data type -->
</Property>
```

3. Data Dictionary Non-Compliant

```
<EntityType Name="ListingProperty"> <!-- Not a Data Dictionary Resource
Name, but it is a Property -->
<Key>
  <PropertyRef Name="ID" />
</Key>
<Property Name="ID" Nullable="false" Type="Edm.Int32"> <!-- should be
ListingKey Data Dictionary Entity Name -->
</Property>
```

4. Data Dictionary Compliant Extension - Entity Set

```
<EntityType Name="Hotsheet">
<Annotation Term="RESO.OData.Metadata.MLSName" />
...
</EntityType>
```

5. Data Dictionary Compliant Extension - declared property

```
<EntityType Name="Property"> <!-- A Data Dictionary Resource Name -->
<Key>
   <PropertyRef Name="ListingKey" />
</Key>
<Property Name="ListingKey" Type="Edm.Int32" Nullable="false" /> <!-- A
Data Dictionary Entity Name -->
<Property Name="DistanceFromVolcano" Type="Edm.Int32" > <!-- NOT a Data
Dictionary Entity Name -->
<Annotation Term="RESO.OData.Metadata.MLSName" />
</Property>
```

6. Data Dictionary Compliant Enumeration

```
<EnumType Name="Country">
  <Member Name="UnitedStates" Value="0"> <!-- Spaces are not allowed in
@Name -->
    <Annotation Term="RESO.OData.Metadata.StandardName" String="US" /> <!--
Accepted/Compliant DD Value -->
    <Annotation Term="RESO.OData.Metadata.MLSName" String="United States"
/> <!-- The MLS Preferred Value -->
    <!-- Open Question: I'm not sure if two <Annotation> elements are
allowed in an EDMX -->
  </Member>
  <Member Name="Canada" Value="1">
    <Annotation Term="RESO.OData.Metadata.StandardName" String="CA" /> <!--
Accepted/Compliant DD Value -->
  </Member>
</EnumType>
```

7. Data Dictionary Enumeration alternate name

```
<EnumType Name="StandardStatus">
  <Member Name="Active" Value="0" /> <!-- Accepted/Compliant DD Value -->
  <Member Name="ActiveUnderContract" Value="1"> <!-- Spaces are not allowed
in @Name -->
    <Annotation Term="RESO.OData.Metadata.StandardName" String="Active
Under Contract" />
  </Member>
</EnumType>
```

8. Data Dictionary MLS specific value

```
<EnumType Name="PropertySubType">
  <Member Name="Apartment" Value="0" /> <!-- Accepted/Compliant DD Value
-->
  <Member Name="BoatSlip" Value="1"> <!-- Spaces are not allowed in @Name
-->
    <Annotation Term="RESO.OData.Metadata.StandardName" String="Boat Slip"
/>
  </Member>
  <Member Name="MlsSpecificPST" Value="1"> <!-- Acceptable Non-DD Value in
Open Enumeration Field -->
    <Annotation Term="RESO.OData.Metadata.MLSName" String="MLS Specific
PropertySubType" />
  </Member>
</EnumType>
```

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 v1.0.3.

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/
HTTP Protocol 1.1	Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing • https://tools.ietf.org/html/rfc7230 Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content • https://tools.ietf.org/html/rfc7231 Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests • https://tools.ietf.org/html/rfc7232 Hypertext Transfer Protocol (HTTP/1.1): Range Requests • https://tools.ietf.org/html/rfc7233 Hypertext Transfer Protocol (HTTP/1.1): Caching • https://tools.ietf.org/html/rfc7234 Hypertext Transfer Protocol (HTTP/1.1): Authentication • https://tools.ietf.org/html/rfc7235
HTTP Protocol 2.0	Hypertext Transfer Protocol Version 2 (HTTP/2) • https://tools.ietf.org/html/rfc7540 HPACK: Header Compression for HTTP/2 • https://tools.ietf.org/html/rfc7541
Transport Layer Security (TLS) (Encryption for HTTP support)	The Transport Layer Security (TLS) Protocol Version 1.2 • https://www.ietf.org/rfc/rfc5246.txt Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) • https://tools.ietf.org/html/rfc7525 OWASP TLS implementation guide • https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet SSL Labs TLS Deployment Best Practices • https://www.ssllabs.com/downloads/SSL_TLS_Deployment_Best_Practices.pdf

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
RCP - WEBAPI-001: Modified: Section 2.4.3	2017-03-22	Sergio Del Rio

RCP - WEBAPI-002: Modified: Appendix 4 - DataSystem XML Schema, 2 - Select a single data system, 3 - How do I look at the metadata for a specific service?	2017-03-22	Sergio Del Rio
RCP - WEBAPI-003: Modified: 2.2 HTTP Protocol, Section 5 - References	2017-03-27	Sergio Del Rio, Geoff Rispin
RCP - WEBAPI-004: Modified: 2.2 HTTP Protocol, Section 5 - References	2017-03-27	Sergio Del Rio, Geoff Rispin
RCP - WEBAPI-005: Modified: 2.6.2 - Replaced Entire Section	2017-08-17	Sergio Del Rio
RCP - WEBAPI-006: Modified: 2.6.1 - Changed MUST to MAY	2017-08-17	Sergio Del Rio
RCP - WEBAPI-007: Modified: 2.4.4 - Removed time row in table	2017-08-17	Sergio Del Rio
RCP - WEBAPI-008: Modified: Many clean-up changes as per the RCP.	2017-08-18	Sergio Del Rio
RCP - WEBAPI-009: Modified 2.4.10 as per the RCP.	2017-08-21	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

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>
```

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",
...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="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
```

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="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="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.DateTimeOffset" 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.DateTimeOffset" 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.DateTimeOffset" 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.DateTimeOffset">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.DateTimeOffset">2014-04-11T11:24:00.6508563-04:00</d:DateTimeStamp>
      <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.DateTimeOffset">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.DateTimeOffset">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.DateTimeOffset">2014-04-11T11:24:00.6508563-04:00</d:DateTimeStamp>
      <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.DateTimeOffset">2014-04-11T11:24:00.6508563-04:00</d:DateTimeStamp>
      <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",
      "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",
      "Localizations": []
   },
   {
      "Name": "Member",
      "ResourcePath": "Member",
      "Description": "RESO Standard Member Resource",
      "DateTimeStamp": "2014-04-11T12:02:48.509401-04:00",
      "Localizations": []
   }],
   "ID": "RESO_MLS"
}]

}
```


Appendix 5 - Approved RCPs

RCPs Approved for Version 1.0.3

RCP - WEBAPI-001 Odata Property Facet Attribute MaxLength, Precision and Scale Errata

Submitter Name	Jeremy Crawford
Submitter Organization	Real Estate Standards Organization
Submitter Email	jeremy@reso.org
Co-submitter Name	Maria Dalarcao
Co-submitter Organization	MLSListings, Inc.
Co-submitter Email	mdalarcao@mlslistings.com

Document Name	RESO Web API v1.0.2
Document Version	1.0.2
Date Submitted	2016-04-13
Status	**IN DRAFT**
Status Change Date	2017-08-17

Synopsis

Clarify the meaning of the attributes of Edm.Decimal and Edm.String in the document RESO Web API v1.0.2 to be compliant with OData Version 4.0.

Rationale

To comply with OData V4 spec, we need to:

- Add explanation to the meaning of Edm.Decimal's attributes Precision and Scale.
- Add the attribute MaxLength to Edm.String.

Proposal

Modify the document RESO Web API v1.0.2, Section 2.4.3 Data Types.

- Add the explanation (bold text below) of the Precision and Scale for decimal data type:

Edm.Decimal — Numeric values with fixed precision and scale.

Precision

- **Is the maximum number of significant digits allowed in the property's value**
- **It MUST be a positive integer**
- **If no value is specified the decimal property has unspecified precision**
- **MUST be a non-negative integer between 0 and 12 for a temporal property**

Scale

- **Is the maximum number of digits allowed to the right of the decimal point**
- **May be a non-negative integer or "variable"**
  - **Integer Value**
    - **The number of digits to the right of the decimal point may vary from 0 to the value of Scale**

- **The number of digits to the left of the decimal point may vary from 1 to the value of (Precision - Scale). If Precision == Scale, then a single 0 must precede the decimal point**
- **Scale must be <= Precision. If no value is specified, Scale defaults to 0**

- **Variable**
  - **The number of digits to the right of the decimal point may vary from zero to the value of the Precision**

- Add MaxLength (bold text below) for string data type:

  Edm.String Sequence of UTF-8 characters

  MaxLength: the maximum length of the string

Implementation example:

<Property Name="AssociationFee" Type="Edm.Decimal" Precision="13" Scale="2" />

<Property Name="BuilderName" Type="Edm.String" MaxLength="50" />

RCP - WEBAPI-002 Remove TimeZoneOffset

Submitter Name	Maria Dalarcao
Submitter Organization	MLSListings, Inc
Submitter Email	mdalarcao@mlslistings.com
Co-submitter Name	
Co-submitter Organization	
Co-submitter Email	

Document Name	RESO Web API v1.0.3
Document Version	1.0.3
Date Submitted	2016-04-13
Status	**IN DRAFT**
Status Change Date	2017-08-17

Synopsis

We are proposing to remove TimeZoneOffset field from the Data System resource collection because DateTimeStamp field contains the time zone offset value in OData V4 and it's not necessary to have TimeZoneOffset field anymore. OData V4 supports only Edm.Date and Edm.DateTimeOffet and DateTime gets converted to DateTimeOffset. In OData V4 Edm.DateTime does not exist.

From OData V4 spec (http://docs.oasis-open.org/odata/odata/v4.0/os/part3-csdl/odata-v4.0-os-part3-csdl.html) OData defines the following two types:

Edm.Date	Date without a time-zone offset
Edm.DateTimeOffset	Date and time with a time-zone offset, no leap seconds

In **RESO Web API v1.0.3** document under **2.6.1 DataSystem Resource,** the Data System Resources Collection defines the following two fields:

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.

Example of medata of DataSystem:

<Property Name="DateTimeStamp" Type="Edm.DateTimeOffset" />

<Property Name="TimeZoneOffset" Type="Edm.String" />

Example of created values:

"DateTimeStamp": "2015-12-01T00:00:00-08:00"

"TimeZoneOffset": "-8"

Rationale

The OData DateTimeStamps carry the time zone as part of the actual datum. This makes the TimeZoneOffset in the Data System superfluous.

Please note that DateTimeStamp's offset portion carries both hours and minutes, e.g. "DateTimeStamp": "2015-12-01T00:00:00***-08:00***".

Proposal

TimeZoneOffset should be removed from Data System Resources Collection (RESO Web API v1.0.3, Section 2.6.1 DataSystem Resource)**.**

The following line should be removed: "TimeZoneOffset The TimeZone Offset provided in standard TimeZone notation of GMT[+|-]X, where X is the number of hours in the offset."

RCP - WEBAPI-003 Update HTTP specification references to current Internet/Industry Standards

Submitter Name	Geoff Rispin
Submitter Organization	Templates 4 Business, Inc.
Submitter Email	grispin@t4bi.com
Co-submitter Name	
Co-submitter Organization	
Co-submitter Email	

Document Name	WebAPI
Document Version	1.02
Date Submitted	2016-05-02
Status	**IN DRAFT**
Status Change Date	2017-08-17

Synopsis

The RESO standards should encourage the adoption of the current predominant internet standards at the time of release. The current WebAPI Document references superseded RFC standards that have been out for some time and the RFC in the RESO documents are now consider obsolete.

The updated standards in include errata, language clarifications, updated security, bug fixes and backwards compatible feature enhancements.

Rationale

The RESO community is building its standards on top of those standards already actively in use on the internet. The RESO standards should reflect that in our documentation by using those standards that are actively in use. Most implementations will already meet to using libraries and frameworks that implement the newer RFCs as developers are using libraries, browsers and other frameworks from sources that have already make these changes years ago.

Proposal

Updates all HTTP RFC references in the WebAPI transport document to their latest specification

- HTTP/1.1 RFC 2616 -> RFC 7230-7237
- HTTP/2 (Not Referenced) -> RFC 7540

Impact

The impact should be minimal as most implementers of the WebAPI transport are using common third party software to implement the HTTP transport within their software. There are very few implementations that are not written to the latest HTTP/1.1 RFC standards (RFC7230-7237). The ones that would be highly impacted are those implementations that wrote their own HTTP stack which should be rare and have taken on a significant technical debt by taking that approach in the first place.

The HTTP/2 includes a separate handshake method that will not impact those that have not implemented it and downgrade gracefully.

Compatibility

The new RFCs are backwards compatible with the old ones except where contradictions or ambiguity exists in the older RFCs.

RCP - WEBAPI-004 Include SSL RFC to ensure secure implementation

Submitter Name	Geoff Rispin
Submitter Organization	Templates 4 Business, Inc.
Submitter Email	grispin@t4bi.com
Co-submitter Name	
Co-submitter Organization	
Co-submitter Email	

Document Name	WebAPI Transport
Document Version	1.0.2
Date Submitted	2016-05-01
Status	**IN DRAFT**
Status Change Date	2017-08-17

Synopsis

The current WebAPI standards document references secure communication with references but does not include the reference standards involved. There are many out of date standards that should not be used as have been "broken" and no longer provide any security. Even with the latest standards, configuration updates must be made for the protocol to be secure as the implementation allows for weak encryption by default that provides no benefit.

Rationale

The current specification does not define any encryption reference for implementation and many of the old implementations that are compatible with HTTP are also cryptographically un-secure or weak. These include all of SSL2 and SSL3 and parts of TLS1.0.

Both Oauth2.0 and OpenID have enforced encryption requirements which have an impact on the WebAPI Transport implementations as well.

Proposal

Add the current standard cryptographic RFCs for HTTPS communication to the specification.

- TLS 1.2 -> RFC5246 https://www.ietf.org/rfc/rfc5246.txt

Add a references to security vendor best practices for the use of Encryption

- RFC 7525 Best Practices - https://www.rfc-editor.org/rfc/rfc7525.txt
- OWASP TLS implementation guide - https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet
- https://www.ssllabs.com/downloads/SSL_TLS_Deployment_Best_Practices.pdf

Include security in the certification process. This step should be a requirement of validating a secure OAuth2 communication for the current specification as strong security is a requirement of the underlying authentication protocol. Extending it to the Transport protocol make sense Users get a false sense of security when they see a secure connection but the implementation is poor or flawed.

- Server test (free service) - https://www.ssllabs.com/ssltest/

Impact

For most implementors it should be minimal as SSL is provided underlying frameworks and libraries that are likely updated. All of the major software implementations (Mozilla NSS, Java SSE, OpenSSL, Microsoft SChannel/CryptAPI, GnuTLS, etc.) provide regular security updates for all issues and implement the latest feature sets. For those that are using out-of-date or un-patched software or custom cryptographic implementations, they will have more work involved to bring them up to a reliable state but their security would be benefiting them little if they do not.

There may be many clients using older libraries that have not been updated they should be encourage to move to non-encrypted communications or update as their implementation are providing no security benefit and making the providers less secure.

Implementations will still need validate their configurations secure their systems properly. The crpytographic software does allow for weak configurations as there are places that they are still valid. This is where the best practices references can help with giving the specification a target for MUST implement and SHOULD implement behaviours.

Compatibility

For those solutions that are maintaining patch levels of their software and frameworks, this should be a transparent endeavor. The compatiblity problem come from the unmaintained solutions and the frequent security updates that happen. Due to the nature of cryptographic vulnerabilities and their announcement, those using un-secure methods would be broken by strict enforcement but those same solutions are have a false sense of security and are gaining no crpytograhic benefits in their implementation.

RCP - WEBAPI-005 Revise Section 2.6.2 - Data Dictionary Resources

Submitter Name	Scott Petronis
Submitter Organization	Onboard Informatics, LLC
Submitter Email	spetronis@onboardinformatics.com
Co-submitter Name	Paul Stusiak
Co-submitter Organization	Falcon Technologies Corporation

Document Name	RESO Web API
Document Version	1.0.3
Date Submitted	2017-03-06
Status	**IN DRAFT**
Status Change Date	2017-08-17

Co-submitter Email	pstusiak@falcontechnologies.com

Synopsis

Section 2.6.2 is revised to provide detail on how Data Dictionary Resources are intended for use in the Web API.

Rationale

Section 2.6.2 is incomplete and does not fully express the intent of the Web API to define limitations and constraints in the use of Odata for the purposes of RESO.

The section currently references empty pages without formal detail. The EDMX files referenced are simply examples and NOT part of the standard definition. Therefore, the section does not currently provide explicit guidance for the standard implementation or assistance in making implementation decisions. Proper EDMX references MAY be made within testing rules or other support documents.

Proposal

The text of section 2.6.2 be changed to

The Web API is intended to facilitate data exchange within a market and across different markets. This makes data sharing more efficient and creating applications that use the data less expensive. To accomplish this, many common field terms, data types and some classes of values are expressed in the RESO Data Dictionary, a separate related standard from RESO.

The Web API is intended, but not restricted, to use resources named in the RESO Data Dictionary. These resources have specific names for certain fields in a resource and specific names for values in certain cases for enumerations.

This section provides guidance to implementors of this standard on how to handle the resources, fields and enumerations to maximize interoperability between implementations.

The standard MAY enforce some or all of these guidance items in the certification testing. Please refer to the Certification Testing Rules for further information.

2.6.2.1 RESOURCE Names

When an implementation has a resource that is substantially similar to a resource name defined in the Data Dictionary, the implementation MUST use the Data Dictionary resource name. Implementations MAY have additional resources that are not defined in the Data Dictionary to meet the needs of the implementation.

2.6.2.2 Entity Names

Within a Data Dictionary resource, when an implementation has a entity (field) name that is substantially similar to a field name defined in the Data Dictionary, the implementation MUST use the Data Dictionary field name for the entity name in the XML and Json representation of the resource. When existing data has not been transformed to match the Data Dictionary data type, an Annotation tag SHOULD be used to indicate that the field is not in compliance with the Data Dictionary and to indicate what the data types is.

Client applications SHOULD inspect entities with respect to data type, and where a data type does not match the expected Data Dictionary value, the application SHOULD expect to find the correct data type in the Annotation tag. Client applications may need special handling to deal with these cases while the industry transitions historical data to new data types.

Within a Data Dictionary resource, many implementations will have entities (fields) that are not part of the Data Dictionary standard. In these cases, implementors and their customers are encouraged to have these entities included in the Data Dictionary. Some entities will remain specific to an implementation based on business rules or other considerations. Entities of this type MUST have an Annotation attached to indicate that this entity is specific to the implementation.

2.6.2.3 Enumeration Entity Names

Within a Data Dictionary resource, when an implementation has an enumeration entity (field) that is substantially similar to a field name and enumeration type in the Data Dictionary, the implementation MUST use the Data Dictionary field name for the enumeration entity.

2.6.2.4 Enumeration Entity Values

Within a Data Dictionary resource, where the entity has a Data Dictionary enumeration entity name, the Data Dictionary may have defined values for enumerations. Where the enumeration value is substantially similar to the Data Dictionary enumeration entity name value, the implementation MUST use the Data Dictionary field name value.

2.6.2.5 Extending Data Dictionary Resources, Names and Values

In many implementations, specific resources, entities and enumeration entity values not part of the Data Dictionary will be required. As

described in 2.6.2.2 for Entity Names, extensions are indicated by an Annotation on the resource, entity or enumeration value.

2.6.2.6 Large Enumerations

Odata in the current and previous versions has a limitation on the size of a multi-value enumeration. Refer to section 2.4.8 Annotations for further details.

2.6.2.7 Examples

1. Data Dictionary Compliant

```
<EntityType Name="Property"> <!-- A Data Dictionary Resource Name -->

<Key>

<PropertyRef Name="ListingKey" />

</Key>

<Property Name="ListingKey" Type="Edm.Int32" Nullable="false" /> <!-- A Data Dictionary Entity Name -->

...
```

2. Data Dictionary Non- compliant (data type)

```
<EntityType Name="Property"> <!-- A Data Dictionary Resource Name -->

<Key>

<PropertyRef Name="ListingKey" />

</Key>

<Property Name="ListingKey" Nullable="false" Type="Edm.String" MaxLength="255"> <!-- A Data Dictionary
Entity Name -->

<Annotation Term="RESO.OData.Metadata.StandardName" String="ListingKey" /> <!-- Non compliant data type
-->

</Property>
```

3. Non-compliant

```
<EntityType Name="ListingProperty"> <!-- Not a Data Dictionary Resource Name, but it is a Property -->

<Key>

<PropertyRef Name="ID" />

</Key>

<Property Name="ID" Nullable="false" Type="Edm.Int32"> <!-- should be ListingKey Data Dictionary Entity
Name -->

</Property>
```

4. Extension - Resource

```
<EntityType Name="Hotsheet">

<Annotation Term="RESO.OData.Metadata.MLSName" />

...

</EntityType>
```

5. Extension - Entity

```
<EntityType Name="Property"> <!-- A Data Dictionary Resource Name -->

<Key>

<PropertyRef Name="ListingKey" />

</Key>

<Property Name="ListingKey" Type="Edm.Int32" Nullable="false" /> <!-- A Data Dictionary Entity Name -->
```

```
<Property Name="DistanceFromVolcano" Type="Edm.Int32" > <!-- NOT a Data Dictionary Entity Name -->
<Annotation Term="RESO.OData.Metadata.MLSName" />
</Property>
```

Impact

This change proposal may impact existing implementations of the Web API.

RCP - WEBAPI-006 Modify 2.6.1 Data System Resource from Must Implement to May Implement

Submitter Name	Scott Petronis
Submitter Organization	Onboard Informatics, Inc.
Submitter Email	spetronis@onboardinformatics.com
Co-submitter Name	
Co-submitter Organization	
Co-submitter Email	

Document Name	RESO Web API v1.0.3
Document Version	1.0.3
Date Submitted	2016-04-13
Status	**IN DRAFT**
Status Change Date	2017-08-17

Synopsis

In the **RESO Web API v1.0.2 specification section 2.6.1** there is reference to a 'DataSystem' resource. The purpose of this resource was to enable organizations to offer client applications a set of pointers to (e.g. URIs) to one or more underlying systems. For example, a service organization that supports multiple MLS clients could use this model to present a single endpoint that contains multiple MLS data resources.

This 'DataSystem' concept has been confused during some implementations and has caused OData compliance tests to fail in some cases. In other cases the 'DataSystem' concept is superfluous as there is no need for pointers to multiple underlying systems.

We are proposing to keep this concept in the standard for those who wish to implement it, but to change it from a **MUST** be supported to a **MAY** be supported. This would still allow organizations to use the concept but would not require organizations to implement it where there is no need. This also keeps with standard OData $metadata usage at the service URI stem.

Appendix 4 would remain intact as an example of how the 'DataSystem' should be implemented for those choosing to use this model.

Specifically, in **RESO Web API v1.0.3** document under **2.6.1 DataSystem Resource,** the following modification would be made:

Modify this "This resource MUST be provided by all implementations."

To this "This resource **MAY** be provided by any implementation."

Rationale

The DataSystem resource acts a a pointer to other underlying systems and resources. In many cases (i.e. servers that support a single MLS) this top level 'DataSystem' is simply not required and creates an additional step in acquiring the underlying metadata. In some cases this may also interfere with standard, off-the-shelf OData client implementations and, therefore, require additional time and cost unnecessarily. For organizations that wish to implement this model, they still may do so, but this should not be required for all organizations.

Proposal

Change the wording in section 2.6.1 of the Reso Web API v1.0.3 from **"MUST be provided"** to **"MAY be provided."**

RCP - WEBAPI-007 Section 2.4.4: Remove required filter function time() (Copy)

Submitter Name	Pace Davis
Submitter Organization	Zillow Group
Submitter Email	pacedavis@big-llc.com
Co-submitter Name	Michael Watt
Co-submitter Organization	Zillow Group
Co-submitter Email	mwatt@big-llc.com

Document Name	RESO Web API v1.0.3 Draft
Document Version	1.0.3
Date Submitted	2017-03-17
Status	**IN DRAFT**
Status Change Date	2017-08-17

Synopsis

This RCP proposes the removal of the required filter function time() in section 2.4.4 of the RESO Web API v1.0.3 Draft document. We submit that servers should not be required to implement this function as there are very few reasonable use cases for it within the context of a real-estate API, and implementing it often has significant performance considerations attached.

Rationale

Section 2.4.4 of the RESO Web API v1.0.3 Draft identifies a number of functions from the OData standard that a RESO Web API implementation must support to be considered compliant. One such function is time(), which extracts the Edm.TimeOfDay component from an Edm.DateTimeOffset value.

We propose the removal of this requirement for the following reasons:

- We can identify no use case for extracting the time component of a Edm.DateTimeOffset in a filter that could reasonably be considered required functionality of a Web API implementation. Consider that the RETS 1.x standards define no equivalent functionality for the DMQL query language.
- Implementations that support filtering on the time component of a date-time value would need to very carefully consider the performance characteristics of such a query. Typical implementations using a relational database may not be able to satisfy such a query using an index, as the time component of a date-time value is not the most significant prefix of the value, resulting in slow full table scans.

Proposal

In section 2.4.4 of RESO Web API v1.0.3 Draft, in the table listing functions that implementations are required to support, remove the following row from the 'Date Functions' sub-section:

time	time(StartTime) le StartOfDay

Further notes

Many of the other functions identified in section 2.4.4 could similarly be argued against. For example, hour(), minute(), second(), and so on. Future RCPs may address the removal of these functions also.

RCP - WEBAPI-008 Web API Version 1.0.2 Specification Errata (Copy)

Submitter Name	Cody Gustafson
Submitter Organization	FBS
Submitter Email	cgustafson@fbsdata.com
Co-submitter Name	Rick Trevino
Co-submitter Organization	MetroList
Co-submitter Email	rtrevino@metrolist.net

Document Name	RESO Web API v1.0.3 Draft
Document Version	1.0.3 DRAFT
Date Submitted	2017-03-21
Date Appended	2017-08-02
Status	**IN DRAFT**
Status Change Date	Status

Co-submitter2 Name	Paul Stusiak
Co-submitter2 Organization	Falcon Technologies
Co-submitter2 Email	pstusiak@falcontechnologies.com

Synopsis

The RESO WEB API v1.0.2 Specification contains several errors. The v1.0.3 document should fix these errors.

Rationale

The working version of the specification should fix errors found in previous versions.

Proposal

The current working version, v1.0.3 should make the following changes:

Correct grammar, spelling errors and broken links.

Update OData 3.0 to OData 4.0 and augment the XML Examples with JSON Examples.

Specific changes:

Section 1.3 Approach - The first paragraph link to the Odata specification is broken.

Section 2 Specification - The first paragraph link to the Odata specification is broken.

Section 2.2.2 X-HTTP-Method-Override Header - The second paragraph, last sentence is missing a closing period.

Section 2.3.2 URI Stem - This section should have JSON examples added. JSON is the primary serialization technique in implementations and all certifications have been done through it.

Section 2.3.2 URI Stem - Standardize the use of 'stem' to be represented as lower case.

Section 2.3.3. Data Systems Endpoint - Paragraph 1: Standardize the use of 'end point' to be two words.

Section 2.3.3. Data Systems Endpoint - Paragraph 3: Correct spelling of 'Pleas' to "Please"

Section 2.4 Search - The first paragraph link to the Odata specification is broken.

~~**Section 2.4.2 Query Support** - First 'Note' block: Fix the word cases of the phrase "Request entity too large"~~ Rejected: RFC 2616 uses capitals on each word of an HTTP response code description.

Section 2.4.3 Data Types - Correct the examples to use RESO Data Dictionary standard names.

‘Price’ should be ‘ListPrice’, ‘ClosePrice’

‘Beds’ should be ‘BedroomsTotal’

‘List Date’, ‘Sale Date’ should be ‘ListingContractDate’, ‘CloseDate’

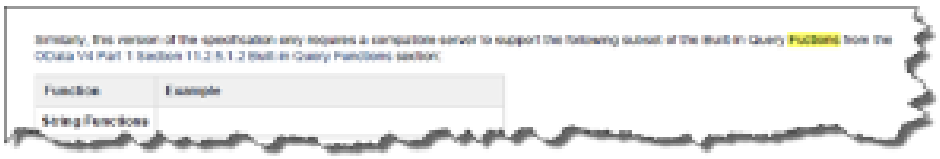
‘Open House Start’ should be ‘OpenHouseDate’

‘Commission’ should be ‘LotSizeAcres’

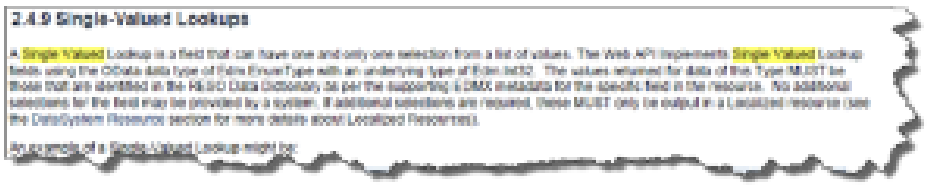
‘Level’ should be ….. ‘Levels’

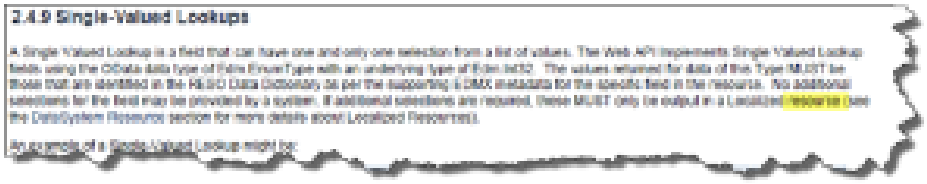
Section 2.4.4 The $filter Option - Correct the spelling of Functions in "Built-In Query Fuctions".

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

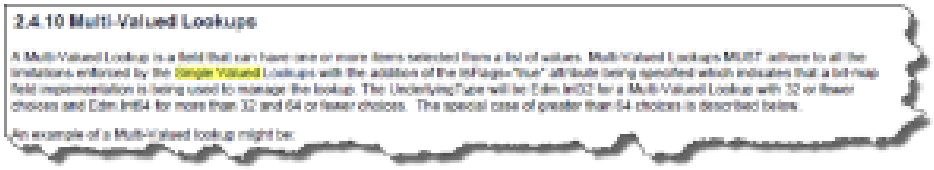


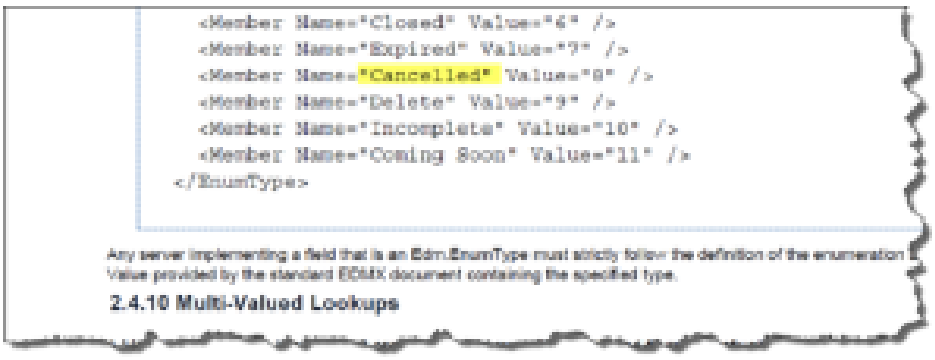
Section 2.4.9 Single-Valued Lookups - Correct use of 'Single-Valued' to be consistent with the rest of the document. Correct the use of 'Resource' to be consistent with the rest of the document.





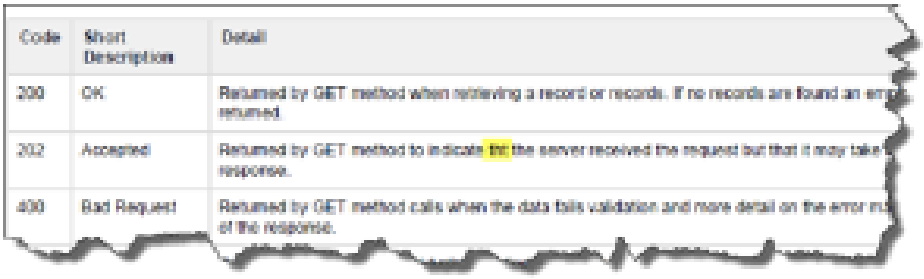
Section 2.4.10 Multi-Valued Lookups - Correct use of 'Single-Valued' to be consistent with the rest of the document. Correct the use of 'Canceled' to be correct.





Section 2.5.3 Error Message Bodies - is referring to OData v3 documentation. So, we need to update that as well as the example to show the correct response format(s). As a side note, it would probably be worth mentioning that a server needs to respond with the format requested with errors.

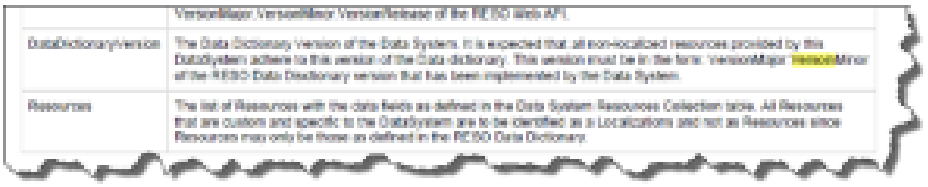
Section 2.5.3 Error Message Bodies - Correct spelling of 'that'

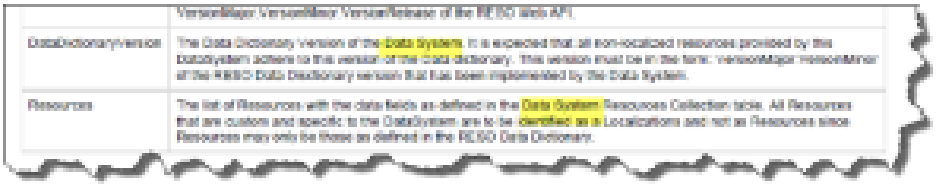


Section 2.5.3 Error Message Bodies - Correct grammar of last paragraph. Correct spelling of 'the'

Section 2.6 Standard Resources The Web API should be independent of specific versions of the Data Dictionary. Remove the references to a specific Data Dictionary versions in the fourth paragraph list. This may be worth skipping in favor of WEBAPI-005

Section 2.6.1 DataSystem Resource - Correct spelling of 'requested'. Correct spelling of 'VersionMinor'. Correct use of 'DataSystem' to be consistent with the rest of the document. Correct URI path to be 'http://odata.reso.org/RESO/Odata/SYS1/Property. Correct spelling of 'Localization'. Correct spelling of 'Description'. Correct URI path to be 'http://odata.reso.org/RESO/Odata/SYS1/Residential.



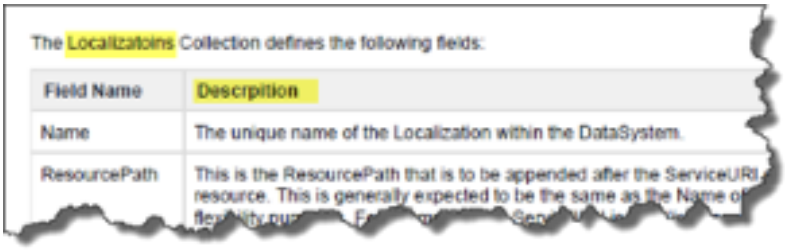


DataDictionaryVersion	The Data Dictionary version of the Data System. It is expected that all non-localized resources provided by this DataSystem adhere to this version of the data dictionary. This version must be in the form: VersionMajor.VersionMinor of the RESO Data Dictionary version that has been 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:

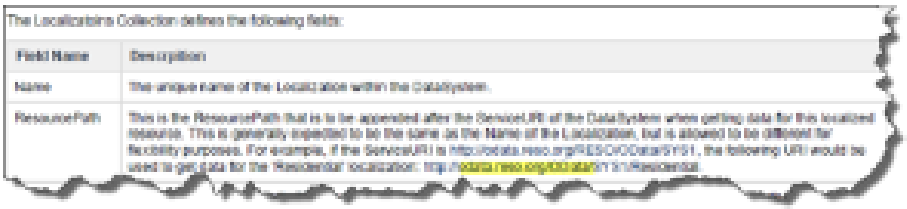
Field Name	Description
Name	The unique name of the Resource within the DataSystem.
ResourcePath	This is the ResourcePath that is to be appended after the ServiceURI of the DataSystem when getting data for that resource. This is generally expected to be the same as the Name of the resource, but is allowed to be different for flexibility purposes. For example, if the ServiceURI is http://odata.reso.org/RESOWOData/DD1DI, the following URI would be used to get data for the 'Property' resource: http://odata.reso.org/DD1DI/DD1DI/Property.
Description	A description of the Resource expected to contain human readable explanations of what data is provided by the resource.

Correct



The Localizations Collection defines the following fields:

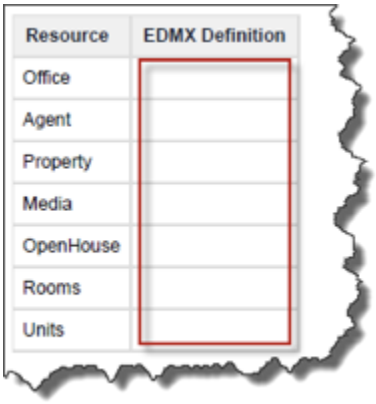
Field Name	Description
Name	The unique name of the Localization within the DataSystem.
ResourcePath	This is the ResourcePath that is to be appended after the ServiceURI resource. This is generally expected to be the same as the Name of



The Localizations Collection defines the following fields:

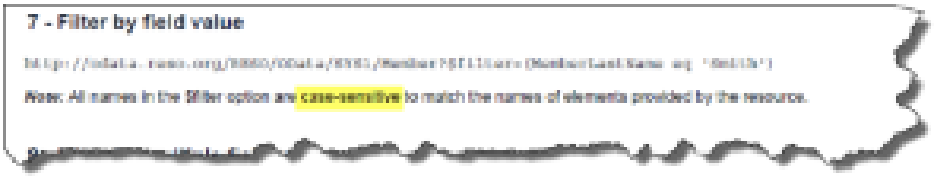
Field Name	Description
Name	The unique name of the Localization within the DataSystem.
ResourcePath	This is the ResourcePath that is to be appended after the ServiceURI of the DataSystem when getting data for this localized resource. This is generally expected to be the same as the Name of the Localization, but is allowed to be different for flexibility purposes. For example, if the ServiceURI is http://odata.reso.org/RESOWOData/DD1DI, the following URI would be used to get data for the 'Residential' localization: http://odata.reso.org/DD1DI/DD1DI/Residential.

2.6.2 Data Dictionary Resources - Missing an EDMX Definition for the Resource in the table.



Resource	EDMX Definition
Office	
Agent	
Property	
Media	
OpenHouse	
Rooms	
Units	

Appendix 2 - Basic Query Examples - Correct use of 'case sensitive' to be consistent with the rest of the document.



7 - Filter by field value

http://odata.reso.org/RESO/OData/DD1DI/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.

Impact

None

Compatibility

None identified

RCP - WEBAPI-009 Collections of Enumerations (Copy)

Submitter Name	Sergio Del Rio
Submitter Organization	Templates for Business, Inc.
Submitter Email	Sergio.Del.Rio@t4bi.com
Co-submitter Name	
Co-submitter Organization	
Co-submitter Email	

Document Name	RESO Web API v1.0.3 Draft
Document Version	1.0.3
Date Submitted	2017-04-24
Status	**IN DRAFT**
Status Change Date	

Synopsis

When version 1.0.2 of the Web API was written, there appeared to be no way to nicely deal with multi-valued-lookups. The team settled on using Enumerations where IsFlags=true which is all we could find at the time that satisfied the problem. However, this limited multi-valued-lookups to 64 items so we added section 2.4.10 Multi-Valued Lookups to the specification to outline a way to implement larger lookups and still use Enumerations where IsFlags=true. Recent research has revealed that both server and client API's support an alternate method which would be to use a Collection of Enumerations. Furthermore, some server vendors have issues with implementing the solution proposed in 2.4.10 due to limitations of the libraries that are available to implement OData on their platforms.

Rationale

This RCP proposes that we add an additional method to implement Multi-Valued Lookups to make it easier for both Clients and Servers to implement than the current only available solution in the specification and also address the implementation issue on some platforms.

Proposal

The solution is to use an Enumeration with IsFlags=false and then, in the field definition define the field as a Collection of the specific Enumeration. This is handled in the metadata quite easily. The following is an example of the metadata for a specific Multi-Valued Lookup:

```
The Enumeration:

<EnumType Name="Association_Amenities" IsFlags="false"
UnderlyingType="Edm.Int64">
<Member Name="Banquet_Facilities" Value="50041194459" /> <Annotation
Term="RESO.OData.Metadata.MRIS.StandardName" <String>Banquet
Facilities</String> </Annotation>
<Member Name="Barbecue" Value="50041194461" />
<Member Name="Biking_Trails" Value="50041194463" /> <Annotation
Term="RESO.OData.Metadata.MRIS.StandardName" <String>Biking
Trails</String> </Annotation>
</EnumType>

The Field:

<Property Name="AssociationAmenities"
Type="Collection(RESO.OData.Metadata.MRIS.Association_Amenities)">
<Annotation Term="RESO.OData.Metadata.MRIS.StandardName">
<String>AssociationAmenities</String> </Annotation>
</Property>
```

With the above metadata, the collection is easily queryable as part of your object as follows:

```
To get records which have only the specified set of amenities:

https://services.dev.mris.com/RESO/OData/MRIS/Property?$format=json&$fil
ter=ListPrice ge 10000 and AssociationAmenities eq
RESO.OData.Metadata.MRIS.Association_Amenities'Biking_Trails,Gated_Parki
ng'&$top=100&$skip=0

To get records which have all of the specified set of amenities but may
have other amenities as well:

https://services.dev.mris.com/RESO/OData/MRIS/Property?$format=json&$fil
ter=ListPrice ge 10000 and AssociationAmenities has
RESO.OData.Metadata.MRIS.Association_Amenities'Biking_Trails,Gated_Parki
ng'&$top=100&$skip=0
```

Proposed Changes to the Specification

Section 2.4.10 should be broken up into two sub-sections:

2.4.10.1 Multi-Valued Lookups - Bitmap Fields

2.4.10.2 Multi-Valued Lookups - Collections of Enumerations

The current text in 2.4.10 should be moved into 2.4.10.1 exactly as it is at this time.

The contents of Section 2.4.10.2 should be as follows:

2.4.10.2 Multi-Valued Lookups - Collections of Enumerations

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. A field that contains Multi-Valued lookups must make use of an Enumeration that has IsFlags=false. The UnderlyingType of the enumeration must be either Edm.Int32 or Edm.Int64 depending on the size of the values that are being returned.

A field that contains Multi-Valued Lookups based on the defined Enumeration must be defined as a Collection of Enumerations.

The examples above will be also included in the section.

Further notes