Web API Core Testing Specification

Data Dictionary Version: 1.6.0+
Web API Core 1.0.2+
August 2020

RESO End User License Agreement (EULA)

This End User License Agreement (the “EULA”) is entered into by and between the Real Estate Standards Organization (“RESO”) and the person or entity (“End User”) that is downloading or otherwise obtaining the product associated with this EULA (“RESO Product”). This EULA governs End Users use of the RESO Product and End User agrees to the terms of this EULA by downloading or otherwise obtaining or using the RESO Product.

Purpose

The goal of the Web API 1.0.2 Core specification is to provide a common, stable set of authentication protocols and API functionality to meet the needs of the real estate industry, with the intent that the Core specification will rarely change going forward.

Endorsements will be used to provide additional functionality to the Core specification in a modular manner and treated as separate specifications with their own dependencies, one of which may or may not be a dependency on Web API Core.

This document will focus exclusively on the Web API Core specification.

Background

The RESO Web API provides an open standard for a RESTful, JSON-based API that's centered around the RESO Data Dictionary, with the ability to support local extension. At its core, the RESO Web API standard is based on a subset of the OData specification from OASIS.

The OData specification provides a) a metadata format, b) a query format and URL structure to support it, c) a response format with a type system. Each of these items MUST be valid with respect to OData for a Web API server to be considered compliant.

Additionally, there are RESO requirements beyond those of OData. For instance, Web API Servers MUST expose at least one *Property, Member, Office, or Media* Data Dictionary resource in order to be certified. There are also authentication requirements, which at the time of writing are that servers MUST support OAuth2 Auth Tokens OR Client Credentials.

The Web API Core testing rules ensure that server metadata are compliant, the data types provided by the RESO Data Dictionary support a minimum set of query operations valid for their types, that the query and response format are correct, and that the results logically match the query that was being used.

Testing Framework

RESO Web API Core certification is provided by the RESO Commander.

The RESO Commander is an open source, cross-platform Java library created by RESO that uses established community libraries, such as the Apache Olingo OData Client, XML parsers, and JSON Schema Validators, to provide a testing API.
Web API tests are written in a high-level testing language (DSL) called Gherkin. This is part of a Behavior Driven Development (BDD) platform called Cucumber, which allows for the expression of testing workflows using a natural language that is intended to be accessible to business analysts, QA testers, and programmers alike.

A command-line interface has been provided during the initial development phase as an entry point into the testing API. This provides the environment used for certification, self-assessment, or can even be run on a test automation server in a continuous integration and deployment platform such as GitHub CI, Jenkins, Travis, or CircleCI to help prevent regressions in a RESO-certified codebase.

A graphical user interface is also available through popular and free Integrated Development Environment (IDE) plugins for IntelliJ and Eclipse. IDEs provide a superior testing platform as they provide better informational messages and are able to run and debug the entire test suite as well as a given test individually. The availability of plugins saves significant time in testing, development, and certification. The level of community support is one of the reasons Cucumber was chosen as a testing platform.

Testing Methodology

Configuring the Test Client

Configuration of the RESO Commander for Web API Certification involves providing a service endpoint, authentication, resource, and field information in a template which will be used during the automated testing process.

A blank Core Web API template exists in the root of the RESO Commander project, and there is also a sample template which is used internally for acceptance testing of the Web API testing tool. The sample template provides a useful reference when filling out RESOScript files. Items marked as REQUIRED in the configuration file must be completed, but things like sample field values have already been provided and should be sufficient for testing. If not, they also may be changed.

Metadata Request Using RESO Standard Authentication

When testing begins, an HTTP request is made to an applicant's given service location with either OAuth2 Bearer Tokens or Client Credentials. Both of these authentication strategies allow for data consumption to be machine automated so that additional interaction from a user isn't necessary during the authentication process. As such, the RESO Data Dictionary Commander can be used for automated testing.

The metadata request is expected to function according to the OData specification in terms of request and response headers and response formats. RESO specifically uses an XML version of OData metadata, which contains an Entity Data Model (EDM) and model definitions, and is often referred to as EDMX.

Metadata Validation

Syntax Checking

Metadata returned from a RESO Web API server are checked for XML validity as well as validated against Entity Data Model (EDM) and EDMX definitions published by OASIS, the creators of the OData specification. If metadata are invalid for any reason, Data Dictionary testing will halt.

Semantic Checking

After metadata syntax has been validated, declared data models are checked for correctness. For example, if a given server declares they support the RESO *Property* resource, then the RESO Commander will look for an OData EntityType definition for *Property*. If the underlying data model is not found, metadata validation will fail with a diagnostic message to help users understand why a given error occurred. Once the model is found, its field and enumeration definitions will be checked for correctness as well.

Another aspect of semantic checking is ensuring that all models have keys so they can be indexed, meaning that a data request can be made to the server by key. This is a basic requirement for fetching data from a server.

RESO Certification

Certification of Web API servers consists mainly of ensuring that a core set of required query operations are supported in a manner adhering to the RESO and OData specifications, and that servers send the appropriate response for each query. These fields are preconfigured in a file, as mentioned in the section on Configuration, rather than sampled from the RESO Data

Dictionary. Data Dictionary resources, fields, and enumerations MUST be used in the configuration of the testing tool for RESO Certification.

In addition to comparison operators, such as greater and less than for things like Integers and Timestamps, OData query operators such as `$select`, which allows the consumer to specify a list of fields to be returned in the payload, or `$top` which allows the consumer to specify the size of the result set. These are outlined in the next section.

Web API Core Testing Requirements

The following table summarizes the testing queries used in the Web API Core specification:

Requirement ID	Description	Section	BDD	Notes
metadata-validation	Request and Validate Server Metadata	2.4.1	Source	See Metadata Validation *REQ-WA103-END3*
service-document	Service Document Request		Source	OData service document request *REQ-WA103-END1*
fetch-by-id	Allows Records to be retrieved by ID.	2.4.1	Source	OData Indexability by Key Requirement *REQ-WA103-QR1*
select	$select Allows fields to be requested on an individual basis as part of a query.	2.4.2	Source	The $select list is what determines the "data shape" for a given query. *REQ-WA103-QR3*
top	$top Allows the client to request a specific number of records in a query.	2.4.2	Source	*REQ-WA103-QR4*

skip	$skip Allows clients to page through records in a contiguous manner, each subsequent page of records continuing where the previous page left off.	2.4.2	Source	*REQ-WA103-QR 5*
orderby-timestamp-asc	$orderby - Allows results to be returned in a specified order.	2.4.4	Source	See here for more info *REQ-WA103-QO 28.1*
orderby-timestamp-desc	$orderby - Allows results to be returned in a specified order.	2.4.4	Source	See here for more info *REQ-WA103-QO 28.2*
orderby-timestamp-asc-filter-int-gt	$orderby - Allows results to be returned in a specified order.	2.4.4	Source	See here for more info *REQ-WA103-QO 28.3*
orderby-timestamp-desc-filter-int-gt	$orderby - Allows results to be returned in a specified order.	2.4.4	Source	See here for more info *REQ-WA103-QO 28.4*
filter-int-and	$filter - Logical Operator: and	2.4.4	Source	See OData 5.1.1.1.7 *REQ-WA103-QO 9*
filter-int-or	$filter - Logical Operator: or	2.4.4	Source	See OData 5.1.1.1.8 *REQ-WA103-QO 10*
filter-int-not	$filter - Logical Operator: not	2.4.4	Source	See OData 5.1.1.1.9 *REQ-WA103-QO 11*

filter-int-eq	$filter - Logical Operator: Integer equal (eq)	2.4.4	Source	See OData 5.1.1.1.1 *REQ-WA103-QO 2*
filter-int-ne	$filter - Logical Operator: Integer not equal (ne)	2.4.4	Source	See OData 5.1.1.1.2 *REQ-WA103-QO 3*
filter-int-gt	$filter - Logical Operator: Integer Greater Than (gt)	2.4.4	Source	See OData 5.1.1.1.3 *REQ-WA103-QO 4*
filter-int-ge	$filter - Logical Operator: Integer greater than or equal (ge)	2.4.4	Source	See OData 5.1.1.1.4 *REQ-WA103-QO5*
filter-int-lt	$filter - Logical Operator: Integer less than (lt)	2.4.4	Source	See OData 5.1.1.1.5 *REQ-WA103-QO 6*
filter-int-le	$filter - Logical Operator: Integer less than or equal (le)	2.4.4	Source	See OData 5.1.1.1.6 *REQ-WA103-QO 7*
filter-decimal-eq	$filter - Logical Operator: Decimal equality (eq)	**TODO**	Source	See OData 5.1.1.1.1
filter-decimal-ne	$filter - Logical Operator: Decimal equality (ne)	**TODO**	Source	See OData 5.1.1.1.2
filter-decimal-gt	$filter - Logical Operator: Decimal equality (gt)	**TODO**	Source	See OData 5.1.1.1.4
filter-decimal-ge	$filter - Logical Operator: Decimal equality (ge)	**TODO**	Source	See OData 5.1.1.1.5

filter-decimal-lt	$filter - Logical Operator: Decimal equality (lt)	**TODO**	Source	See OData 5.1.1.1.3
filter-decimal-le	$filter - Logical Operator: Decimal equality (le)	**TODO**	Source	See OData 5.1.1.1.4
filter-date-eq	$filter: `DateField eq 'yyyy-mm-dd'`	**TODO**	Source	See OData 5.1.1.6.1 Support date part of ISO-8601 date.
filter-date-ne	$filter: `DateField ne 'yyyy-mm-dd'`	**TODO**	Source	See OData 5.1.1.6.1 Support date part of ISO-8601 date.
filter-date-gt	$filter: `DateField gt 'yyyy-mm-dd'`	2.4.4	Source	See OData 5.1.1.6.1 *REQ-WA103-QO 25* Support date part of ISO-8601 date.
filter-date-ge	$filter: `DateField ge 'yyyy-mm-dd'`	**TODO**	Source	See OData 5.1.1.6.1 Support date part of ISO-8601 date.
filter-date-lt	$filter: `DateField lt 'yyyy-mm-dd'`	**TODO**	Source	See OData 5.1.1.6.1 Support date part of ISO-8601 date.
filter-date-le	$filter: `DateField le 'yyyy-mm-dd'`	**TODO**	Source	See OData 5.1.1.6.1 Support date part of ISO-8601 date.

filter-datetime-eq	Equal to DateTimeOffset	**TODO**	Source	
filter-datetime-ne	Not equal to DateTimeOffset	**TODO**	Source	
filter-datetime-gt	Greater than DateTimeOffset	2.4.4	Source	**TODO**
filter-datetime-ge	Greater than or equal to DateTimeOffset	2.4.4	Source	**TODO**
filter-datetime-lt	$filter: Less than given DateTimeOffset	2.4.4	Source	See OData 5.1.1.6.11 *REQ-WA103-QO 29*
filter-datetime-le	$filter: `TimestampField le DateTimeOffset`	2.4.4	Source	See OData 5.1.1.6.9 *REQ-WA103-QO27* Support datetime for ISO-8601 date.
filter-enum-single-has	`has` operator for Edm.EnumType	2.4.9	Source	See here for more info *REQ-WA103-QM 7*
filter-enum-single-eq	`eq` operator for Edm.EnumType	2.4.4	Source	**TODO** *REQ-WA103-QO 12*
filter-enum-single-ne	`ne` operator for Edm.EnumType	2.4.4.	Source	**TODO** *REQ-WA-103-QO 13*
filter-enum-multi-has	`has` operator for Edm.EnumType and IsFlags=true	2.4.9	Source	See here for more info *REQ-WA103-QM 8*
filter-enum-multi-has-and	Multi-Value lookups has operator with and	2.4.9	Source	See here for more info *REQ-WA103-QM 8.2*

filter-coll-enum-any	`any()` lambda for Collection of Edm.Enumtype	2.4.9	Source	See OData 5.1.1.10.1 *REQ-WA103-QM 3*
filter-coll-enum-all	`all()` lambda for Collection of Edm.Enumtype	2.4.9	Source	See OData 5.1.1.10.2 *REQ-WA103-QM 4*
response-code-400	Tests that server supports 400 responses	???	Source	
response-code-404	Tests that server supports 404 responses	???	Source	

Sample Queries

Sample queries for the above requests can be found in the RESO Web API Core Walkthrough.

Certification Workflow

The Certification workflow has been optimized around self-assessment prior to certification.

Application

Those seeking RESO Certification will apply with the Membership Department prior to having their application reviewed by Certification. Once an application has been processed, testing will begin, starting with a review of the OData request, response, and metadata format. For more information, see the section on metadata validation.

Self Assessment

It's expected that vendors will use the RESO Web API testing tool before applying for certification in order to ensure they will be able to pass.

Guides exist to help them with the evaluation process (TODO: guide).

Any questions regarding certification should be directed to certification@reso.org.

Certification Review

A RESOScript file is required for review. This file should contain credentials and the service location of the Web API Server to be tested, as well as configuration parameters outlined in the Configuration section.

Certification Issuance

Once RESO Certification verifies that a given configuration produces no errors, unhandled warnings, or exceptions, certification will be granted and the applicant will be issued a certification report. Once the vendor has reviewed this certification, the certification report will be posted on the public RESO website.

Feature Requests

Feature requests can be requested as issues on the RESO Commander's GitHub project, or by contacting josh@reso.org or certification@reso.org.

Support

Certification Support will be provided by certification@reso.org.

Tool support is provided by josh@reso.org.

Contributors

Thanks to the following contributors for their help with this project:
Sergio Del Rio
Eric Finlay
Steve Ledwith
Paul Stusiak
Certification Subgroup Attendees

If you would like to contribute, please contact josh@reso.org or certification@reso.org. This could mean anything from QA or beta testing to technical writing to doing code reviews or writing code.

Written by Josh Darnell.