Orange Button API Specification

Released

May 1, 2017

ABSTRACT

This document specifies requirements for a standardized web services API for Orange Button™ data exchange. It is intended as a model for software providers serving the solar finance industry.

Change History

D-1	February 24, 2017	Initial draft.
Final	May 1, 2017	Incorporated review comments.

Contents

1 Introduction

The objective of this document is to specify a set of requirements for the Orange Button™ web services Application Program Interface (API) that can be used for data exchange with systems supporting data elements as specified in Orange Button™ specifications and data taxonomy. This specification identifies core interface principles that can be implemented across systems that may have different architectures, but support the Orange Button™ data taxonomy. The current version of the Orange Button™ specification and taxonomy is available to SunSpec oSDX workgroup members only from SunSpec.org. Contact membership@sunspec.org if you would like to join this work group.

Each section in the document contains a conceptual description of the interface functionality and a specific set of requirements that pertain to that functionality. The requirements have been placed in a dedicated section identified as requirements.

The following definitions are used for the statements in each requirements section:

"**shall**" – Requirement that must be implemented.

"**should**" – Desired but not mandatory.

"**may**" – Permitted but not required.

2 Web Service API

The Representational State Transfer (REST) concept was originally defined by Roy Fielding[1]. The basic elements of a RESTful interface originate from this definition and the initial concept continues to drive the core definition. However, some additional variations are used in different implementations.

This web service API is specified as an HTTP-based RESTful interface.

RESTful web services have become a standard method of interacting with systems to exchange data. The RESTful methodology is used in this specification as the basis for the specific web service requirements. Many best-practice guides exist for the implementation of a REST web services API. The requirements specified in this document attempt to reflect accepted industry standard practices.

The web service requirements in this document are based on an interface using Hypertext Transfer Protocol (HTTP) requests. The foundation of the interface is based on the representation of system resources specified in the network Uniform Resource Locator (URL) and the HTTP method of the request. The URL identifies the resource that is being accessed and the HTTP method specifies the action that is

[1] Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures.* Doctoral dissertation, University of California, Irvine, 2000.

applied to the resource.

2.1 Requirements

The web service interface shall be implemented using a RESTful HTTP-based client/server model as defined in the requirement sections within this document.

The web service interface shall comply with the HTTP/1.1[2] specification including the use of status codes.

2.2 Resources

Resources are the logical elements in the system that can be accessed through the API. A request URL specifies the resource that is being addressed in the request. The resource hierarchy is used in the request to identify the specific resource. Resources can be accessed individually or as a collection. A resource id is used to specify a resource within a resource collection. A resource hierarchy is specified in a URL in the general form:

```
<base URL>/resources/resource id/resources/resource id
```

The examples in this section use a simple two-level resource hierarchy consisting of organizations, and plants. The system contains multiple organizations. An organization can contain one or more plants. The following are URL examples to address resources in the system.

Specify the collection of all the organizations:

```
<base URL>/organizations
```

Specify a specific organization:

```
<base URL>/organizations/:organization_id
```

Specify the collection of all the plants in an organization:

```
<base URL>/organizations/:organization_id/plants
```

Specify a specific plant:

```
<base URL>/organizations/:organization_id/plants/:plant_id
```

2.2.1 Requirements

A resource shall always be represented in the plural form in the URL even in the case of access to a single instance of the resource.

The full resource hierarchy shall be specified in the URL when accessing resources.

[2] Hypertext Transfer Protocol (HTTP/1.1) was updated in 2014 by IETF RFC 7230 (https://tools.ietf.org/pdf/rfc7230.pdf) and family (RFCs 7231-7235)

2.3 HTTP Methods

The HTTP method containing the web service request indicates the action to be performed on the specified resource. The standard HTTP methods used in a REST implementation are: GET, PUT, POST, and DELETE.

2.3.1 GET

The GET method performs the read function for a resource or collection of resources.

When applied to a collection, all the members of the collection are returned. The following URL would return all the plants in the organization with the id of "3":

```
GET <base URL>/organizations/3/plants
```

When applied to a single resource, the data elements associated with the resource are returned. The following URL would return the data contents for plant with the id of "10" in the organization with the id of "3":

```
GET <base URL>/organizations/3/plants/10
```

URL parameters can be used to filter the resources and resource content that is returned.

The GET method does not alter the state of any resources.

2.3.2 PUT

The PUT method performs the update function for a resource or collection of resources.

When applied to a collection, the complete collection is replaced with the contents of the PUT request. The following URL would replace all the plant resources for organization "3" with the contents of the data contained in the POST request.

```
PUT <base URL>/organizations/3/plants
```

When applied to a single resource, the data elements associated with the resource are replaced with the contents of the data contained in the POST request. The following URL would replace the data contents for the plant with the id of "10" in the organization with the id of "3":

```
PUT <base URL>/organizations/3/plants/10
```

2.3.3 POST

The POST method performs the create function for a resource within a collection.

When applied to a collection, a new resource is created in the collection containing the contents contained in the POST request. The following URL would create a new plant resource in the organization "3" plants collection:

```
POST <base URL>/organizations/3/plants
```

The POST method is not valid for a resource and should return a “Method not allowed” (405) HTTP error response.

2.3.4 DELETE

The DELETE method performs the delete function for a resource or collection of resources.

When applied to a collection, the complete contents of the collection are deleted. The following URL would delete all the plant resources for organization “3”:

```
DELETE <base URL>/organizations/3/plants
```

When applied to a single resource, the specified resource is deleted from the collection. The following URL would delete the plant with the id of “10” in the organization with the id of “3”:

```
DELETE <base URL>/organizations/3/plants/10
```

2.3.5 Requirements

The GET method for a resource collection shall return the set of resources in the collection. The depth of information in the response for each resource should be determined by query parameters specified in the request. A subset of resources should be returned based on query parameters in the request.

The GET method for a resource shall return the contents of the resource. The depth of information in the response for each resource should be determined by query parameters specified in the request.

The GET method shall not alter the state of any resource in the system.

The PUT method for a resource collection shall replace the complete resource collection with the contents of the PUT request.

The PUT method for a resource shall replace the contents of the resource with the contents of the PUT request.

The POST method for a resource collection shall create a new member of the collection containing the contents of the POST request.

The POST method for a resource should return a “Method not allowed” (405) HTTP error response.

The DELETE method for a resource collection should delete all the resources in the resource collection.

The DELETE method for a resource should delete the specified resource from the

resource collection.

2.4 Content Type

Web service requests and responses can contain data associated with the operation. The most common data encodings used in REST interfaces for general data transfer are JSON and XML.

2.4.1 Requirements

Content type support shall include JSON or XML.

Content type support should include JSON and XML.

JSON content shall use the HTTP *Content-Type* header set to *application/json*.

XML content should use the HTTP *Content-Type* header set to *application/xml*.

XML content shall use the HTTP *Content-Type* header set to *application/xml* or *text/xml*.

Other encodings may be permitted for specific operations but JSON or XML shall be available for standard resource operations.

The content types supported shall be indicated in the HTTP *Accept* header.

2.5 Error Reporting

The status of a request is indicated by the HTTP status code as outlined in the HTTP specification. The web service API uses HTTP status codes to indicate success or failure and the reason for failure observing the categorization of client and server errors in the HTTP specification.

Additional error information can be included in the body of the response to give more information about the error. It is useful to return both an error code that can be used programmatically as well as a human readable message.

2.5.1 Requirements

HTTP status codes shall indicate the status of the request.

The request response should contain additional error detail including an error code and error message.

2.6 Security

Security of the resources in the system are a critical consideration. It is important that both access and transfer of information be secured.

2.7 Authentication/Authorization

Access to resources in the system is controlled by authentication of the requestor and verification of authorization to access the requested resource.

Open Authorization (OAUTH) 2.0[3] is a widely used industry standard protocol for authorization. Due to the complexity of security implementation and maintenance, it is important to adopt a robust and well-maintained framework. OAUTH 2.0 has a growing number of system implementers and available software packages. For these reasons, OAUTH 2.0 is used for system authorization in this API standard.

2.7.1 Secure Transport

The standard method of transport security for an HTTP-based system is the use of HTTPS/Transport Layer Security (TLS). This is the mechanism that is used in this API specification.

2.7.2 Requirements

OAUTH 2.0 shall be supported for system authorization of requests.

HTTPS/TLS version 1.2[4] or greater shall be used to access resources through the API.

3 Testing Tools and Certification

Testing tools and a certification program for the Orange Button™ API specification are in development and scheduled for release in 2Q 2018. Please contact certification@sunspec.org if you are interested in participating in this program.

[3] See https://oauth.net/2/

[4] Transport Layer Security Protocol Version 1.2 - https://tools.ietf.org/html/rfc5246