HOBOlink® Web Services V2 Developer's Guide

Onset Computer Corporation
470 MacArthur Blvd.
Bourne, MA 02532
www.onsetcomp.com

Mailing Address:
P.O. Box 3450
Pocasset, MA 02559-3450

Phone: 1-800-LOGGERS (1-800-564-4377) or 508-759-9500
Fax: 508-759-9100
Support: http://www.onsetcomp.com/support/contact

Technical Support Hours: 8AM to 8PM ET, Monday through Friday
Customer Service Hours: 8AM to 5PM ET, Monday through Friday

17382-E

Contents

Section 1: Overview

HOBOlink web services offer the ability for third-party developers and partners to write their own programs to extract HOBO® logger data (U30, RX3000, HOBOmobile) from the HOBOlink database.

HOBOlink exposes a set of REST web services hosted on Onset's remote servers. These web services are accessed through a third party software program (the "caller"). In addition, you can also access the deprecated Onset SOAP web services for a limited time, covered in the last section.

The REST web services deliver customized datasets to the caller in several formats (Excel, CSV, HOBOwareCSV, and JSON). This relies on HOBOlink's underlying infrastructure where data is stored as individual readings, such that custom datasets can be extracted and sent to the caller.

Specific capabilities of HOBOlink REST web services include:

- Data from the logger is stored as individual entries in a data warehouse type of database rather than only in binary .dtf files. This provides a tremendous amount of extensibility for allowing customized access to a user's data, both public and private.
- Datasets returned to the caller can span multiple devices, launches, and wraps, as well as being constrained to only a small subset of a deployment.
- Datasets are customizable by time, device S/N, sensor S/N, and/or measurement type.
- Authentication is achieved via a token (provided by Onset) that is passed to HOBOlink as part of the web services call. HOBOlink manages access to the various web services using these tokens. Tokens will be provided free to customers who purchase HOBO devices.

If you wish to discuss the HOBO Remote Monitoring System and web services in more detail, please contact an Onset Computer Application Specialist at (800) 564-4377 or sales@onsetcomp.com.

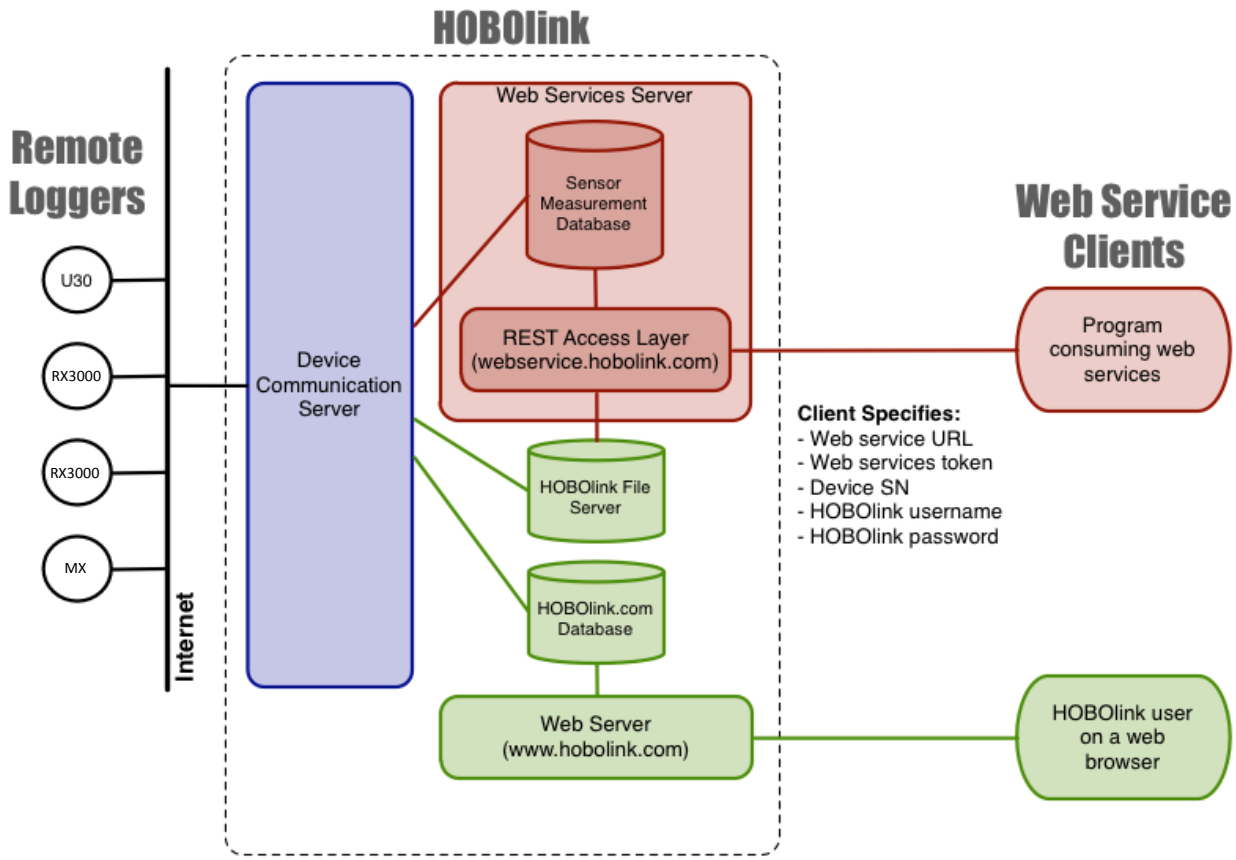
Requirements

- You should be a software engineer with a strong working knowledge of your programming language, its abilities, and its limitations.
- Ideally, you should have experience writing code that consumes REST web services.
- You should have a HOBOlink account and a Remote Monitoring Station and/or a MX series logger, or plans to purchase one in the near future.

Interaction between Web Services, Clients, and Devices

The following diagram shows the interaction between HOBOlink web services, clients, and devices, including:

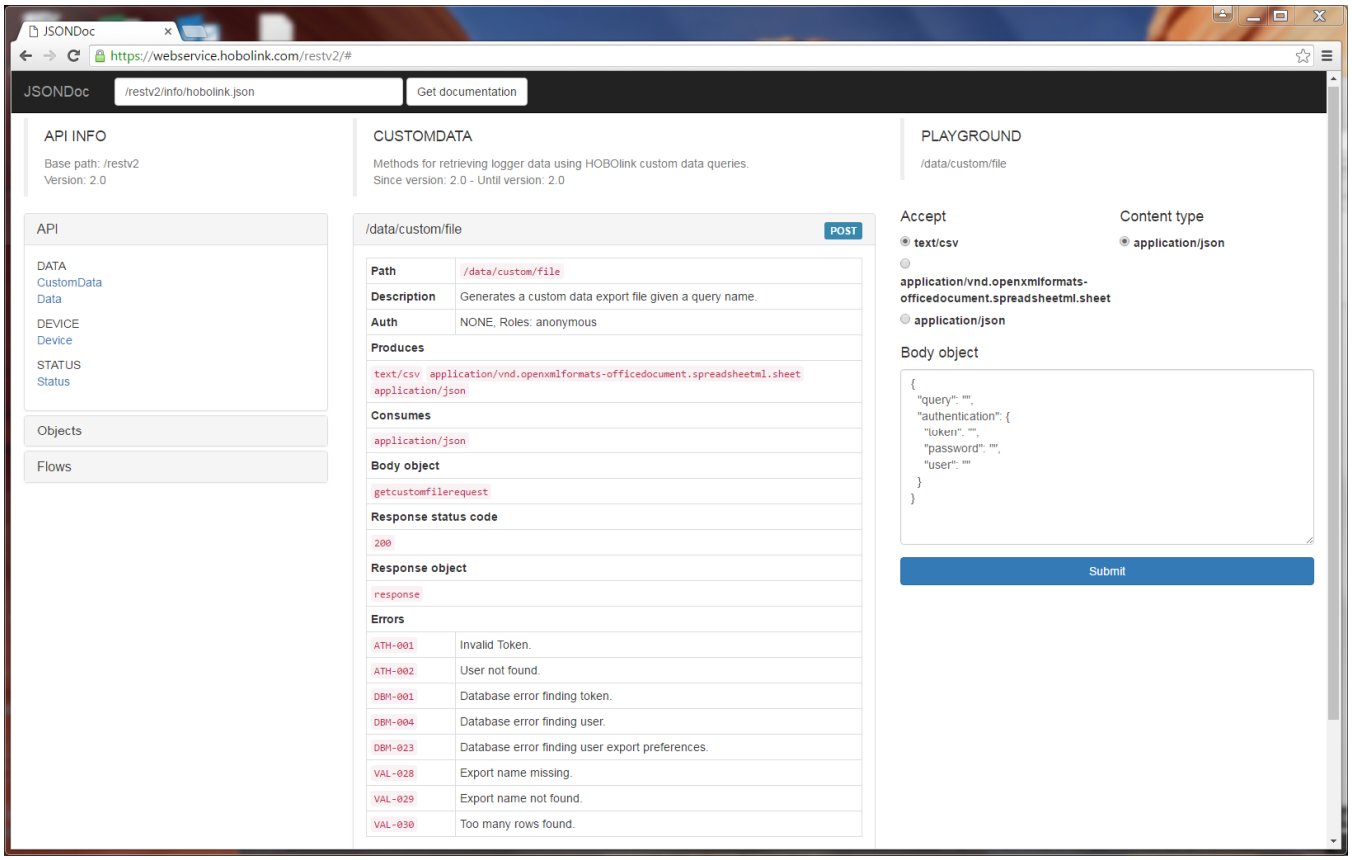
- The difference between a user hitting HOBOlink and a program hitting the web services.
- How the web services get data from the database, not directly from the devices.
- The parameters the web service program needs to specify.

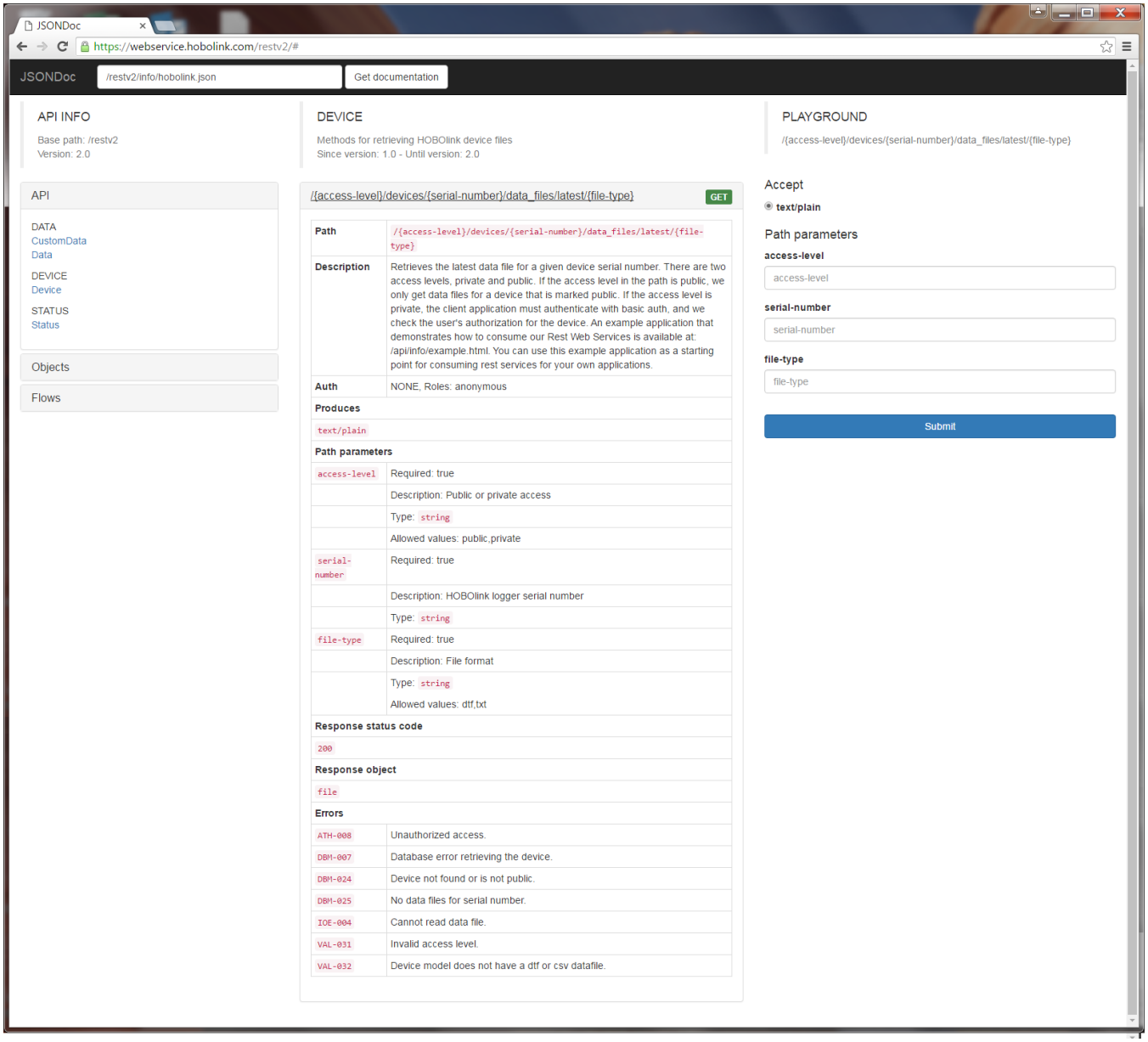


Section 2: REST Web Services Tutorial

Onset's REST web services allow you to easily obtain the latest data and date files from both public and private loggers and incorporate them into your application. This section explains the URLs for obtaining the data using the REST web services.

The REST web services are described in a JSONDoc website shown in the following two screen shots. To access this site, go to https://webservice.hobolink.com/restv2. The JSONDoc site provides a playground section that allows you to interact with the web services and your data without writing any code.



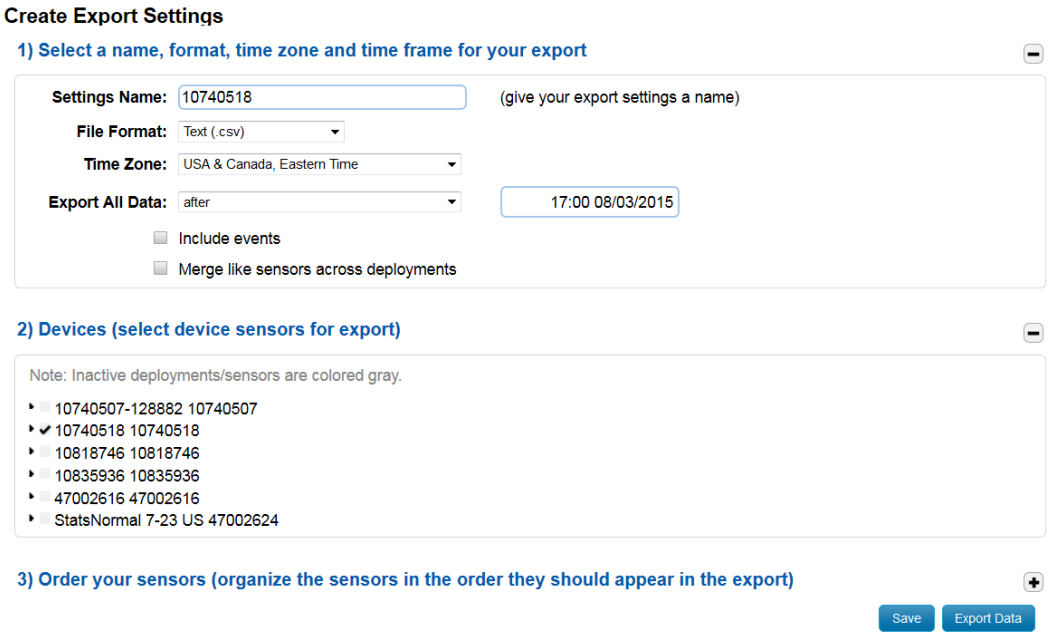


Getting Data using a HOBOlink Custom Data Export

To use the CustomData web service you must set up a custom data export in HOBOlink. To do this:

1. Click Data (and then the Custom Data tab if visible) and then click the Create Export Settings button.

2. Complete the details in all three sections on the Create Export Settings page below (see HOBOlink Help for details).



3. Save the export and then click Export Data to ensure the data results are as expected.
4. Once the export has the data you want, test it using the JSONDoc site shown at the beginning of this section using your token, user info, and query (export).
5. Use the example code in Section 3 to call the web service programmatically.

Getting the Latest Data File for a U30 Device

If a device has been made public in HOBOlink (via HOBOlink User Settings), anyone can access its data via the public URL at http://webservice.hobolink.com/restv2/public/devices. Public services can be executed without authentication or the need to use SSL.

If a device has not been made public, its data can only be accessed via a private URL, which is http://webservice.hobolink.com/restv2/private/devices. Private services require authentication and must be executed over SSL. The authentication scheme is Basic Authentication over SSL (secure sockets layer). An authenticated user can only access data for devices that the user has registered in his or her HOBOlink account, or have been made public.

The data file service public URL follows this convention:

http://webservice.hobolink.com/restv2/public/devices/<serial_number>/data_files/latest/<data_file_type: dtf or txt>

and the data file service private URL follows this convention:

http://webservice.hobolink.com/restv2/private/devices/<serial_number>/data_files/latest/<data_file_type: dtf or txt>

where <serial_number> is the device serial number, and <data_file_type: dtf or txt> is either dtf (to get the dtf file) or txt (to get the text file).

See the JSONDoc site Device API for more details and to test the web service.

Example URLS

Getting the latest .dtf file for a public device with serial number 123456:
http://webservice.hobolink.com/restv2/public/devices/123456/data_files/latest/dtf

Getting the latest .txt file for a public device with serial number 123456:
http://webservice.hobolink.com/restv2/public/devices/123456/data_files/latest/txt

Getting the latest .dtf file for a private device with serial number 123456:

https://webservice.hobolink.com/restv2/private/devices/123456/data_files/latest/dtf

Getting the latest .txt file for a private device with serial number 123456:

https://webservice.hobolink.com/restv2/private/devices/123456/data_files/latest/txt

Testing the Code

The URLs in the previous section point to a stable server with production user devices. It is strongly recommended that you get your code running, tested, and debugged against the HOBOlink development environment before switching to a stable server. To use the development instance of our REST web services for testing, use this URL for public devices:

http://webservice-dev.hobolink.com/restv2/public/devices/1216485/data_files/latest/dtf

and this URL for private devices:

https://webservice-dev.hobolink.com/restv2/private/devices/1216485/data_files/latest/dtf

Section 3: Sample REST Code

This section describes an example application that demonstrates how to consume the REST web services using Java. If you plan to write your client in Java, you can use this example application as a starting point for consuming REST web services for your own applications. If you plan to write your client in another language, refer to documentation provided by your API that describes how to consume REST web services in that language. Onset does not provide support for writing your own web service clients outside of what is contained in this document.

Overview

You can access Onset's REST web services using any language you choose, provided you have an understanding of how to access HTTP content in that language. While the example client provided is written in Java, many other mainstream languages, such as Perl and .NET, have libraries you can use to consume RESTful web services.

The example uses a Java library called the Jersey Client API for RESTful Web Services. For information about the Jersey Client API, visit https://jersey.dev.java.net/ or search for "Consuming REST Web Services with Jersey" on the web. You may also use other Java libraries, such as Apache HttpClient at http://hc.apache.org/httpclient-3.x/.

To download the source code for this example application, go to:
https://webservice.hobolink.com/restv2/info/ExampleRestWebServicesClient.zip

What the Example Application Does

The example code calls the custom data file web service to run the given data query (export) and download the resulting files in text or Excel format. After the example application retrieves the files, it saves them to the local file system.

Also, for U30 devices only, the example calls the private Onset REST device web service to get the latest data files for a given device's serial number in two formats. The first file retrieved from the service in the example is a binary .dtf file and the second is in text format (.csv). After the example application retrieves the files, it saves them to the local file system. This example also demonstrates how to authenticate over SSL (secure sockets layer) to the private Onset REST web services.

Note: There is also a public device web service that does not require authentication or SSL. To use this public service, skip the steps for SSL and authentication. The use of the client web service is the same. The only requirement for accessing a device's data from the public service is that the device has to have been made public. To make a device public, log onto HOBOlink, click the User Settings tab, and check the appropriate box under Preferences > By Device.

Running the Example Java Application

Requirements:

- Maven2 to build the project. Check the version of Maven installed by typing **mvn -version** on the command line. If you don't already have maven installed, visit http://maven.apache.org.
- Java 1.5 or later. Check the version of Java installed by typing **java –version** on the command line.

Build the Source and Run

1. Download the source at https://webservice.hobolink.com/restv2/info/ExampleRestWebServicesClient.zip
2. Extract the zip file to any location.
3. Open the file src/main/java/com/onset/test/rest/webservices/OnsetRestWebServiceExample.java in an IDE (NetBeans, Eclipse, IntelliJ, etc.), or import the pom.xml file.
4. Run the following Maven command:
   mvn compile
5. Run the main class in the IDE. You should see "success" printed out on the console.

Section 4: Reference

Error Codes

Configuration

- CFG-001 – Error loading logging properties file.
- CFG-002 – Error loading the webservice properties file.

Database

- DBM-001 – Database error validating token.
- DBM-002 – Database error retrieving communication plans by VAR.
- DBM-003 – Database error retrieving HOBOlink status.
- DBM-004 – Database error retrieving the user.
- DBM-005 – No active deployments for user.
- DBM-006 – Database error retrieving active deployments for user.
- DBM-007 – Database error retrieving the device by serial number.
- DBM-008 – Database error retrieving the communications plan.
- DBM-009 – Database error saving the device.
- DBM-010 – Database error retrieving the deployment by device.
- DBM-011 – No valid communication plans for the respective VAR.
- DBM-012 – No subscribers found.
- DBM-013 – Database error retrieving list of subscribers.
- DBM-014 – Database error retrieving subscriber.
- DBM-015 – Database error retrieving credit type.
- DBM-016 – Database error saving subscriber.
- DBM-017 – Database error retrieving subscriber's webservice.
- DBM-018 – Database error retrieving webservice.
- DBM-019 – Database error saving subscriber's webservice.
- DBM-020 – No text files for deployment.
- DBM-021 – Database error retrieving latest text file for deployment.
- DBM-022 – Logger communications plan is terminated.
- DBM-023 – Database error retrieving user's export preferences.
- DBM-024 – Device not found or is not public.
- DBM-025 – No data files for serial number.

Authentication

- ATH-001 – Invalid token.
- ATH-002 – Invalid user.

- ATH-003 – Encryption algorithm error during authentication.
- ATH-004 – Invalid password.
- ATH-005 – Invalid device serial number.
- ATH-006 – Invalid communications plan.
- ATH-007 – Invalid status.
- ATH-008 – Device not owned by respective user.

Email

- EML-001 – Error sending plan renewal confirmation email.

I/O

- IOE-001 – Could not read text file.
- IOE-002 – Error communicating with Sensor Observation Service.
- IOE-003 – The Sensor Observation Service reported an error.
- IOE-004 – Cannot read data file.

Validation

- VAL-001 – Time period start time is null.
- VAL-002 – Time period start time is in the future.
- VAL-003 – No data found in specified time range.
- VAL-004 – Device serial number or user name is required.
- VAL-005 – Unsupported preset time period.
- VAL-006 – Unsupported indeterminate value for time period.
- VAL-007 – A start time is required for time period with the "during" operator.
- VAL-008 – An end time is required for time period with the "during" operator.
- VAL-009 – A start time is required for time period with the "after" operator.
- VAL-010 – An end time is required for time period with the "before" operator.
- VAL-011 – Invalid time period operator.
- VAL-012 – An operator is required with a time period.
- VAL-013 – Sensor information is required with "greater than" filter operator.
- VAL-014 – The value is required with "greater than" filter operator.
- VAL-015 – Sensor information is required with "less than" filter operator.
- VAL-016 – The value is required with the "less than" filter operator.
- VAL-017 – Sensor information is required with "greater than or equal to" filter operator.
- VAL-018 – The value is required with the "greater than or equal to" filter operator.
- VAL-019 – Sensor information is required with "less than or equal to" filter operator.
- VAL-020 – The value is required with the "less than or equal to" filter operator.

- VAL-021 – Sensor information is required with "equal to" filter operator.
- VAL-022 – The value is required with the "equal to" filter operator.
- VAL-023 – Sensor information is required with "not equal to" filter operator.
- VAL-024 – The value is required with the "not equal to" filter operator.
- VAL-025 – Invalid filter operator.
- VAL-026 – An operator is required with a filter.
- VAL-027 – Request is null.
- VAL-028 – Invalid data query.
- VAL-029 – Data query not found.
- VAL-030 – Too many rows found for query.
- VAL-031 – Invalid access level.
- VAL-032 – Device model does not have a .dtf or .csv data file.

Glossary

Bind	The process of representing the information in an XML document as an object in computer memory.
Endpoint	A web service endpoint is a (referenceable) entity, processor, or resource to which web service messages can be addressed.
Marshal	The act of converting an XML document to and from a programming object.
REST	Representative State Transfer. An architectural style that allows for point-to-point communication over HTTP using XML.
Sensor Observation Service	Provides an API for managing deployed sensors and retrieving sensor data and specifically "observation" data.
SOAP	Simple Object Access Protocol. A protocol for exchanging XML-based messages over computer networks.
Stub	Also known as skeleton code. A method of generating class files based off a web service definition file (WSDL). Most programming languages have SOAP tools designed to generate stub code from a WSDL file (e.g. wsdl2java).
WSDL	Web Services Description Language. Describes the web service interface. Typically used to develop web service clients.

Section 5: SOAP Web Services Reference

IMPORTANT: New implementations of SOAP web services have been discontinued as of March 2016. This section is included for reference only for existing SOAP web services users.

Step-by-Step Instructions

1. Get the Code.

   The web service's methods are revealed through the WSDL file published by Onset at:

   https://webservice.hobolink.com/axis2/services/HOBOlinkAPIV2?wsdl

   The WSDL is an XML representation of the web service's interface. The same WSDL is used for our test and production environments.

   NOTE: The WSDL may contain other web service offerings. You should not attempt to call these other services.

   Most programming languages and/or IDEs have tools available to help with this conversion.

   Run the program appropriate to your environment that converts the WSDL to usable code. This will vary depending on your programming language.

   The previous API documented in earlier versions of this guide is available if needed. Contact Onset for more information.

Tips	
Java	• We recommend using axis2 to generate Java code from the WSDL: Java - http://axis.apache.org In particular, check out the documentation that explains the command line tool and available plug-ins: http://axis.apache.org/axis2/java/core/docs/reference.html http://axis.apache.org/axis2/java/core/tools/index.html • Let your IDE do the work for you. If there is a plug-in for your IDE listed on the axis2 tools page or in the documentation for your IDE, install, and run it. We have successfully used both NetBeans and IntelliJ to create test clients for our web services. • Local java classes will be created that represent the web service interface.
C#	• See: C# - http://msdn.microsoft.com/en-us/library/7h3ystb6.aspx
PHP4	• See: PHP4 - http://www.nusphere.com/php_script/nusoap.htm
PHP5	• See: PHP5 - http://sourceforge.net/projects/wsdl2php/
Ruby	• See: Ruby - http://www.tutorialspoint.com/ruby/ruby_web_services.htm

2. Contact Onset for Required Information.

   Before you run client code, contact Onset Sales or Technical Support (1-800-LOGGERS) to obtain the necessary authentication and validation information. You will be provided with:

   - A device serial number and HOBOlink username/password, for use in testing
   - One generic validation token for use in testing

- One individualized token you can use in your production environment with your purchased devices.

3. Write Client Code.

   Referencing the web service interface generated in Step 1, write client code that calls the available methods. Your code needs to specify the web service URL, HOBOlink username, password, token, device serial number(s), and sensor serial number(s). Other details such as time period and other filters are also available through the interface code. It is up to you whether you want to use a time period or other data filters when calling the web service.

   The top-most method of the Sensor Observation Service is *getObservationFull*, which takes a query object as its argument. You will populate the query with the details listed above, before passing it to *getObservationFull.* The details of this and all the objects contained in this web service are defined in the Sensor Observation Service section in Section 3.

   In your test code, use the test values provided by Onset along with the following endpoint URL:

   https://webservice-dev.hobolink.com/axis2/services/HOBOlinkAPIV2

   The test account should help you to get your basic framework up & running. You will not be able to run your code until you finish Step 4: Configure Client for SSL.

   Once your tests are working properly, modify your client code to request data from your device serial number and sensors, using your own HOBOlink account information and your individualized validation token. Make sure to also remove the port (:89) from your web service URL, which will allow you to call into our production environment.

4. Configure Client for SSL.

   Getting your client code to connect to an SSL URL will vary depending on your code. Generally speaking, you will need to add the certificate from the web service server to a trusted certificate file, and then reference that from within your code.

5. Write SensorML Parsing Code.

   Data will be returned to the caller in SensorML format. If necessary, Onset can provide sample Java data binding code to marshal SensorML into Java objects.

Guidelines

- Bad or missing sensors will record -888.88 as their reading. You should screen for this value.
- The maximum number of the measurements that will be returned in a single call to the getObservationRequest service is 20,000.
- There is a small chance that duplicate entries can exist for the same sensor at the same time. You should ensure that your code can handle duplicate measurements.
- Observations are returned to the caller in UTC, not the local time of the device.
- The web services do not use the special XML markup CDATA to encode the SensorML in the SOAP response. Some programming languages may need to do additional processing to unescape this response. In our testing, the Java and C# languages needed no additional processing.

Sample SOAP Code

Java

```
import com.onset.aurora.webservices.GetObservationFullDocument;
import com.onset.aurora.webservices.GetObservationFullResponseDocument;
import com.onset.aurora.webservices.HOBOlinkAPIV2Stub;
import com.onset.aurora.webservices.beans.xsd.Authentication;
import com.onset.aurora.webservices.beans.xsd.ObservationQueryFull;
import com.onset.aurora.webservices.beans.xsd.TemporalFilter;
import com.onset.aurora.webservices.beans.xsd.TimePeriod;
import org.apache.axis2.AxisFault;

import java.net.URL;
import java.util.Calendar;

// This example uses the Axis2 WSDL2Code Maven plugin to generate Java client
// stubs from the HOBOlink Web Services WSDL file located at:
//
//     https://webservice.hobolink.com/axis2/services/HOBOlinkAPIV2?wsdl
//
public class GetObservationFullExample {

    // The cert file needs to contain the certificate at
https://webservice.hobolink.com
    final private static String CERT_FILE = "src/test/config/jssecacerts";

    // The service is defined by its endpoint:
    // dev:    https://webservice-dev.hobolink.com/axis2/services/HOBOlinkAPIV2
    // stable: https://webservice.hobolink.com/axis2/services/HOBOlinkAPIV2
    final private static String WS_URL = "https://webservice-
dev.hobolink.com/axis2/services/HOBOlinkAPIV2";";

    final private static String TOKEN = "1vHXNvx1QC";
    final private static String USER = "MRDTester";
    final private static String PASSWORD = "onset123";


    public static void main(String args[]) {
        // This line is required for SSL to work.
        System.setProperty("javax.net.ssl.trustStore", CERT_FILE);

        try {
            HOBOlinkAPIV2Stub stub = new HOBOlinkAPIV2Stub(WS_URL);

            // Set up the request with serial numbers, authentication object,
            // time period, and other filters
            GetObservationFullDocument reqDoc =
GetObservationFullDocument.Factory.newInstance();
            GetObservationFullDocument.GetObservationFull req =
reqDoc.addNewGetObservationFull();

            // Setup authentication parameter
            req.addNewAuth();
            req.getAuth().setToken(TOKEN);
            req.getAuth().setUser(USER);
            req.getAuth().setPassword(PASSWORD);

            // Setup query parameter
            req.addNewQuery();

            // Serial number filter
            req.getQuery().addSerialNumbers("1216485");
```

```
            // Time period filter
            req.getQuery().addNewTimePeriod();
            req.getQuery().getTimePeriod().setOperator("after");
            req.getQuery().getTimePeriod().addNewTime();
            req.getQuery().getTimePeriod().getTime().setStart("2013-12-
01T00:00:00+00:00");

            System.out.println("Request: " + reqDoc);
            // Execute web service
            GetObservationFullResponseDocument response =
stub.getObservationFull(reqDoc);
            System.out.println("Response: " +
response.getGetObservationFullResponse().getReturn());

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

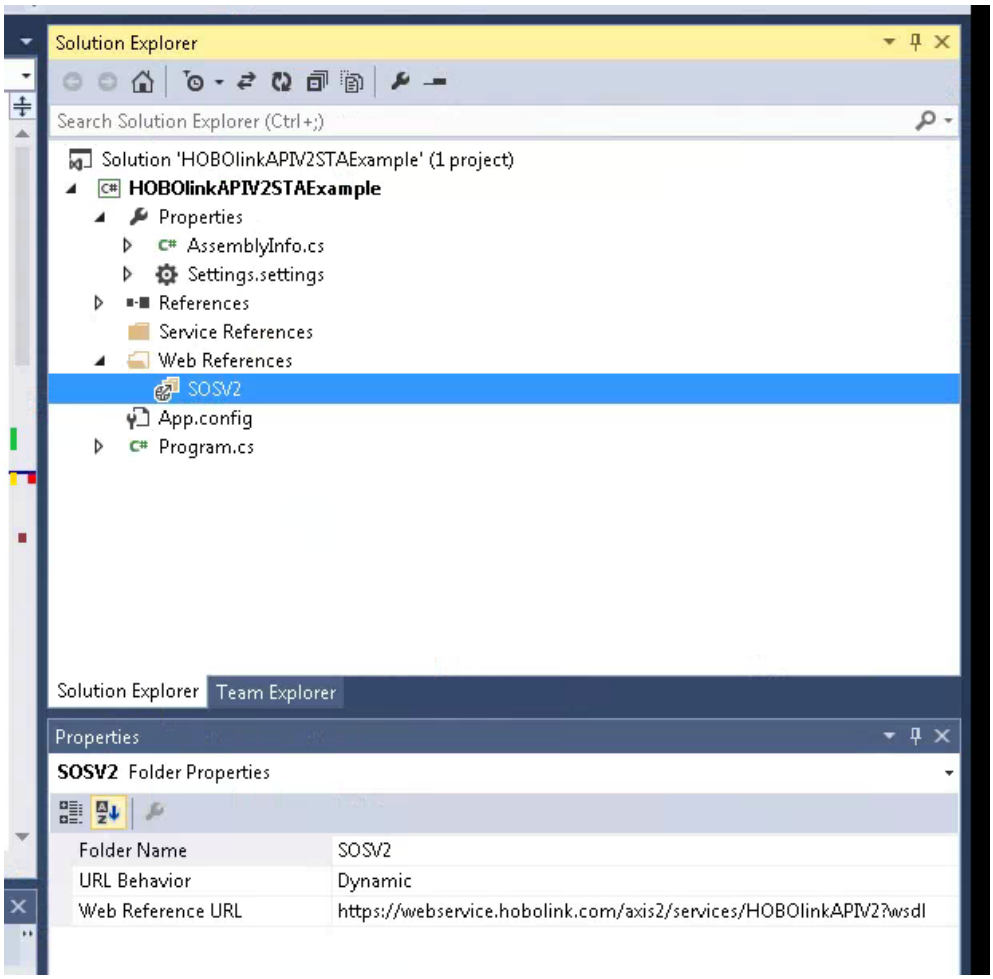
.NET/C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using HOBOlinkAPIV2STAExample.SOSV2;

namespace HOBOlinkAPIV2STAExample
{
    class Program
    {
        static void Main(string[] args)
        {
            HOBOlinkAPIV2 stub = new HOBOlinkAPIV2();

            Authentication auth = new Authentication();
            ObservationQueryFull query = new ObservationQueryFull();

            auth.token = "your token";
            auth.user = "your user";
            auth.password = "your password";

            string[] serialNumbers = { "your logger serial numbers" };
            query.serialNumbers = serialNumbers;

            query.timePeriod = new TemporalFilter();
            //query.timePeriod.preset = "Past_24";
            query.timePeriod.@operator = "During";
            //query.timePeriod.@operator = "After";

            query.timePeriod.time = new TimePeriod();
            query.timePeriod.time.start = "2014-12-10T13:00:00+00:00";
            query.timePeriod.time.end = "2014-12-10T14:00:00+00:00";

            query.sensors = new SimpleSensor[1];
            query.sensors[0] = new SimpleSensor();
            query.sensors[0].sensorSerialNumber = "logger SN:sensor SN";
```

```
            try
            {
                String result = stub.getObservationFull(auth, query);
                Console.WriteLine("The result is: " + result);
                Console.WriteLine("done");

                System.IO.File.WriteAllText(@"ws-response.xml", result);
            }
            catch (Exception e)
            {
                Console.WriteLine("Error: " + e);
            }
        }
    }
}
```

Example Project

Set up a Web Reference for the HOBOlink web service as shown below:



PHP4, PHP5, Ruby - Not available at this time

Sensor Observation Service

- The WSDL can be found at https://webservice.hobolink.com/axis2/services/HOBOlinkAPIV2?wsdl
- The Sensor Observation Service endpoint is located at https://webservice.hobolink.com/axis2/services/HOBOlinkAPIV2
- getObservationFull (ObservationQueryFull query)
- Returns: String of Sensor Observation Service/O&M xml

Parameters

Hierarchy

The following illustration shows the relationship between parameters.

query **ObservationQueryFull**
- userName
- serialNumbers
- timePeriod **TemporalFilter**
  - operator
  - time **TimePeriod**
    - start
    - end
    - startIndet
    - endIndet
  - preset
- sensors **SimpleSensor**
  - sensorSerialNumber
  - measurementType
- filters **ComparisonFilter**
  - operator
  - sensor **SimpleSensor**
    - sensorSerialNumber
    - measurementType
    - value
    - value upper

auth **Authentication**
- token
- user
- password

Parameter Information

Name	Description
query	The object defining the query itself. Required: Y - Type: ObservationQueryFull
• userName	HOBOlink account user. If this parameter is populated, data will be returned for all devices that have been registered by that user. The serial_numbers parameter does not need to be populated. Either a list of serial number(s) or a user is required. Required: N - Type: String
• serialNumbers	Serial numbers of the U30 devices. If this parameter is populated, data will be returned for the specified devices. The user parameter does not need to be populated. Either a list of serial number(s) or a user is required. Required: N - Type: Array of String Objects
• timePeriod	Programming object that represents a time based filter of the data set. Required: N - Type: TemporalFilter
○ operator	The type of time based filter. Supported values are "Before", "After" and "During". Required: Y Type: String
○ time	Time Period for which the data is desired. Required: Y - Type: TimePeriod
▪ start	Start time in ISO 8601 format for the data to be returned. Required for the After and During operators. Required: N - Type: String
▪ end	End time in ISO 8601 format for the data to be returned. Required for the Before and During operators. Required: N - Type: String
▪ startIndet	Can be used in lieu of the start parameter to represent an indeterminate time. Supported value is "Now". Required: N - Type: String
▪ endIndet	Can be used in lieu of the end parameter to represent an indeterminate time. Supported value is "Now". Required: N - Type: String
○ preset	Use to preset the time rather than using a TimePeriod object. Valid arguments are: "Past_24", "Past_1", "Since_Midnight" Required: N - Type: String
• **sensors**	The list of sensors to be returned in the data set. Required: N - Type: Array of SimpleSensor objects
▪ sensorSerialNumber	Serial number of the sensor. Required: Y - Type: String
▪ measurementType	Measurement type of the sensor. Required: Y - Type: String
• **filters**	Programming object that represents a comparison based filter of the data set. Required: N - Type: Array of ComparisonFilter objects
○ ComparisonFilter	
▪ operator	The type of comparison based filter. Supported values are "PropertyIsEqualTo", "PropertyIsNotEqualTo", "PropertyIsLessThan", "PropertyIsGreaterThan", "PropertyIsLessThanOrEqualTo" or "PropertyIsGreaterThanOrEqualTo". Required: Y - Type: String
▪ sensor	Programming object that represents a sensor. Required: Y - Type: SimpleSensor

Name	Description
– sensorSerialNumber	Serial number of the sensor. Required: Y - Type: String
– measurementType	Measurement type of the sensor. Required: Y - Type: String
▪ value	The value to be filtered upon. Required: Y - Type: String
▪ value upper	Required: Y - Type: String
Authentication	Required: Y - Type: Object
• token	Authenticates the caller as a valid CDS Full consumer. Contact Onset Computer for a free token. SSL must be used to ensure encryption of this information. Required: Y - Type: String
• user	HOBOlink account under which the specified serial number is registered. Required: N - Type: String
• password	Password for the specified HOBOlink account. SSL must be used to ensure encryption of this information. Required: N - Type: String

Consuming Web Services

Java - http://axis.apache.org

C# - http://msdn.microsoft.com/en-us/library/7h3ystb6.aspx

PHP4 - http://www.nusphere.com/php_script/nusoap.htm

PHP5 - http://sourceforge.net/projects/wsdl2php/

Ruby - http://www.tutorialspoint.com/ruby/ruby_web_services.htm

SensorML

http://www.opengeospatial.org/standards/sensorml

XML Data Binding

Onset has successfully generated data binding code with the Java toolset XMLBeans (http://xmlbeans.apache.org/). The SensorML schema was too complex for basic Java tools like Castor or JAXB. To date, we have not investigated any XML data binding tools for other programming languages.