AVEVA™ Communication Drivers Pack – Standards - WEBSVC Driver

User Guide

Publication date: Tuesday, May 9, 2023

Publication ID: 868896

Contact Information

AVEVA Group Limited
High Cross
Madingley Road
Cambridge
CB3 0HB. UK

https://sw.aveva.com/

For information on how to contact sales and customer training, see https://sw.aveva.com/contact.

For information on how to contact technical support, see https://sw.aveva.com/support.

To access the AVEVA Knowledge and Support center, visit https://softwaresupport.aveva.com.

Contents

Chapter 1

Getting Started with WEBSVC Communication Driver

WEBSVC Communication Driver - Terminologies

Reference the following terminologies to understand the functioning of the WEBSVC Communication Driver.

SOAP

Simple Object Access Protocol (SOAP) is a XML-based web services access protocol that allows programs that run on disparate operating systems (such as Windows and Linux) to communicate over an HTTP network. For more information about SOAP, see

REST

Representational State Transfer (REST) is a web standards based architecture using the HTTP protocol for data communication. It is used to build web services (RESTful service) that are lightweight, maintainable, and scalable. The data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs).

URL/URI

A Uniform Resource Locater (URL) or a Uniform Resource Identifier (URI) is a string of characters (typically, link on the web) that identifies a resource for interaction over a network (World Wide Web or intranet) using specific protocols.

WSDL

Web Services Description Language (WSDL) is an XML-based interface definition language that describes the functionality offered by a web service. WSDL URL is the web link associated with the web service. The web address (or URL) of the web service that supports SOAP connection ends with **wsdl.** This is used to retrieve the structure of WSDL of the web service.

Web Service Address

Web Service Address is the base address in the primary storage (or main memory) that serves as a reference point for other addresses and memory locations.

Authentication

Authentication is the process in which the credentials provided are compared to those on file in a database of authorized user information on a local operating system or within an authentication server. If the credentials match, the process is completed and the user is granted authorization for access.

The REST interface of the WEBSVC Communication Driver supports basic authentication and OAuth 2.0 (bearer-token) authentication. Different web services support different authentication methods.

- **Basic Authentication:** Basic authentication relies on a user-provided username and a password. The Communication Driver client sends its stored user name and password to the web service server. If authenticated, the connection and communication are established. This authentication type is supported by more applications and is often simpler to use.
- **OAuth 2.0 Authentication:** OAuth 2.0 is an open authorization protocol which enables applications to obtain limited access to user accounts on an HTTP/S service, by means of an authentication server.

Note: The WEBSVC Communication Driver does not support the authentication for SOAP-based web services at the time of this release.

About SOAP and REST Connections

The protocols used to access web services are

- Simple Object Access Protocol (SOAP)
- Representational State Transfer (REST)

The support for SOAP or REST protocol depends on which connection the web services support. For SOAP- based web services, the WEBSVC Communication Driver is configured to support SOAP protocol. For REST-based web services, the WEBSVC Communication Driver is configured to support REST protocol.

The differences between SOAP and REST connections are:

SOAP	REST
• SOAP-compliant web services allow clients to invoke web services and receive responses independent of language and platforms. • SOAP is independent of a specific transport type. • SOAP supports distributed communication. • SOAP relies exclusively on XML to provide messaging services. • SOAP supports multiple instances. • SOAP relies exclusively on the Web Service Description Language (WSDL) for its interactions.	• REST-compliant web services allow clients to access data using a uniform and predefined set of operations. • REST requires HTTP for transport. • REST supports communication between only two endpoints. • REST does not exclusively rely on XML to provide a response. The WEBSVC Communication Driver client supports both JSON and XML based interactions. • REST does not support multiple instances. The instance name is always "WW".

You can create multiple SOAP or REST configurations to connect with multiple web servers.

Setting up the WEBSVC Communication Driver for the First Time

If you are setting up the WEBSVC Communication Driver for the first time, perform the following tasks in the order listed:

1. In the OI Server Manager tree, under the Local node, navigate the WEBSVC Communication Driver in the Operations Control Management Console (OCMC). See Navigating to the WEBSVC Communication Driver.
2. Configure the global parameters. See Configuring Global Parameters in the Communication Drivers Pack Help.

   Note: The default value of the **Buffered Data (Maximum Queued Updates)** is 100 for the WEBSVC Communication Driver.
3. Add a PORT connection.
4. Add a SOAP or a REST connection. Select the connection type based on the Web Service technology that the web service supports.
5. Add one more device groups.
6. Add device items.
7. Activate the Communication Driver. See Activating/Deactivating the OI Server in the Communication Drivers Pack Help.
8. Troubleshoot any problems. See Troubleshooting the WEBSVC Communication Driver.

Chapter 2

Introduction to the Web Service (WEBSVC) Communication Driver

About the WEBSVC Communication Driver

The Web Service Communication Driver (WEBSVC) is an information exchange system, that allows programs and applications using SOAP and REST web services to communicate with each other over the network. It supports programs and applications built using different technologies, running on different platforms and frameworks.

The WEBSVC Communication Driver subscribes to data from web service(s) on the internet or intranet, and operates in stand-alone mode. It provides the common functionality and behavior of the Communication Drivers Pack, including multi-instance and multi-version support.

Supported Client Protocols

The client applications connect to the WEBSVC Communication Driver using the following protocols:

- OPC
- SuiteLink
- DDE/FastDDE
- PCS

For more information, refer to the "Support Client Protocols" section in the Communication Drivers Pack Help.

The WEBSVC Communication Driver communicates messages with SOAP or REST connections over an HTTP network. For more information about SOAP and REST, see About SOAP and REST Connections.

Licensing for WEBSVC Communication Driver

The WEBSVC Communication Driver supports the activation-based licensing to acquire the license both locally and remotely.

For more information on activation-based licensing, see "Centralized (Activation-Based) Licensing" in the Communication Drivers Pack Help.

Navigating to the WEBSVC Communication Driver

You can access the WEBSVC Communication Driver from the OI Server Manager hierarchy within the OCMC. When installed, the WEBSVC Communication Driver appears as a node in the OI Server Manager.

Chapter 3

Configuring the WEBSVC Communication Driver

Adding a PORT Connection

The server-specific configuration portion of the WEBSVC Communication Driver hierarchy tree under the OI Server Manager starts at the PORT Connection object. This object lets you set server parameters for communication with agents (devices) in the hierarchy tree (SOAP/REST).

By default, the installation of the WEBSVC Communication Driver has the port pre-configured. This step is only necessary if the port has been consequently deleted.

To add a PORT Connection object

1. In the console tree, right-click **Configuration** and then click **Add PORT_WEBSVC Connection**. The **New_PORT_WEBSVC_000** object appears in the hierarchy.
2. Edit the object name to appropriately describe components of your specific hardware environment. If you do not rename the object at this time, a numeric sequencing system is applied. You can rename the hierarchy entry later.

Note: The PORT_WEBSVC is the network card in the PC which communicates with the WEBSVC Communication Driver. No configuration is necessary for this node.

Adding and Configuring a SOAP Connection

You can add the SOAP connection from the PORT configuration object. The SOAP node object tests the WEBSVC connection at configuration and at run time.

Adding a SOAP Connection

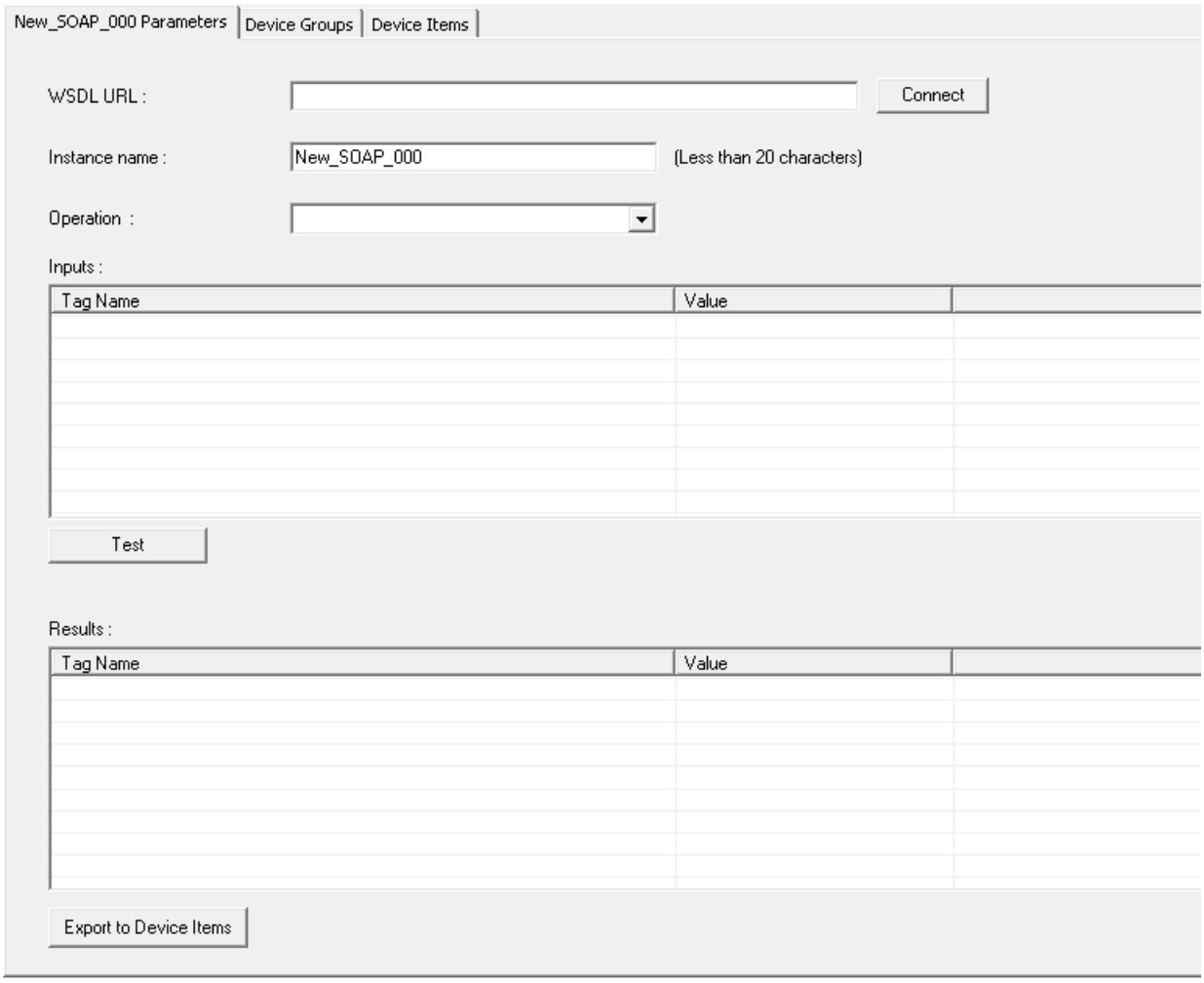
To add a SOAP connection to your WEBSVC hierarchy

1. Right-click on the **New_PORT_WEBSVC_000** object, and select **Add SOAP Connection**.

   The **New_SOAP_000** object is created.

2. Rename the newly created connection as appropriate. The **New_SOAP_000 Parameters** view is displayed.

Configuring a SOAP Connection



To connect to a web service

1. In the **WSDL URL** field, enter the address of the web service you want to connect.

The WSDL URL can only contain a maximum of 255 characters. For more information about WSDL, see WEBSVC Communication Driver - Terminologies.

2. Click **Connect**.

   The **Connection** dialog displays the status of the connection to web service.

3. If the connection to the web service is successful, click **OK**.

   The **Operation** drop-down list displays the operations/methods that can be performed for the web service being invoked.

   The **Inputs** grid populates the tag names based on the selected operation, and is not editable. For more information about tag syntax, see Tag Naming Conventions.

To invoke a web service

1. In the **Instance Name** field, enter the instance of the SOAP connection you want to invoke. By default, the name of the currently active SOAP node is displayed. For more information on naming instances, see Tag Naming Conventions.
2. From the **Operation** drop-down list, select the operation to be performed. The options differ based on the web service you invoke.
3. To enter a value to execute the operation, or to modify an existing value, point to the **Inputs** grid, double-click the **Value** column for the corresponding tag name, and enter the new value.

   Note: When testing the SOAP-based web service connection, ensure to enter values for all the input parameters in the **Value** column of the **Inputs** grid. Otherwise, the test operation will fail.

4. Click **Test.**

   This allows the user to test the connection to the web service and also examine the results received.

   a. Test Response Time

      Upon completion of test execution, the **Test Response Time** displays the time taken (in ms) by the web service to respond and the item count. Use the result to configure the minimum update interval of a topic. For more information, see Device Group Definitions.

   b. Polling the web service - Poll-based/trigger-based (on-demand) polling

      To trigger the web service on demand, use the **.run** tag from the **Tag Name** column in the **Results** grid. Alternatively, you can poll using the update interval for device groups. See Device Group Definitions.

      For more information about different types of polling, see Triggering a Web Request.

   c. Retrieving the Results

      The **Results** grid displays the resulting tags and values, and is not editable. The list of tags in the **Tag Name** column are the items available for subscription in run time. Each tag name is the equivalent of a PLC register, that represents a reference to be used by the application. The result sets received from the web service are in XML format, which are parsed by the WEBSVC Communication Driver and the individual parameters are exposed as reference.

   d. Exporting the Results

      To export results to **Device Items** tab, click **Export to Device Items.**The tags in the **Results** grid are exported to the **Device Items** tab, and the respective item references are populated at run time

Example - Test SOAP-based Web Service for Simple Math Calculations

1. In the **WSDL URL** field, enter the following web service address - http://www.dneonline.com/calculator.asmx?wsdl and click **Connect**.

   The **Connection** dialog displays the status of the connection to web service. Upon the confirmation of a successful connection, click **OK**.

2. In the **Instance Name** field, enter the instance of the SOAP connection. By default, it displays the name of the current active SOAP object.
3. From the **Operation** drop-down list, select an operation - **add/divide/multiply/subtract**. Each operation uses two variables - x and y.
   - **add** operation - the value of the x variable is added to the value of the y variable.
   - **divide** operation - the value of the x variable is divided by the value of the y variable.
   - **multiply** operation - the value of the x variable is multiplied by the value of the y variable.
   - **subtract** operation - the value of the y variable is subtracted from the value of the x variable.
4. In the **Inputs** grid, enter the values of x and y to be used for calculation.
5. Click **Test**.

   The resulting values are used for the values of the custom tag.

   For instance, for an add operation for values x=3 and y=4, the result value 7 is displayed in the **Results** grid.

Adding and Configuring a REST Connection

You can add the REST connection from the PORT configuration object. The REST node object tests the WEBSVC connection at configuration and at run time.

Adding a REST Connection

To add a REST connection to your WEBSVC hierarchy

1. Select and right-click on the **New_PORT_WEBSVC_000** object, and select **Add REST Connection**.

   The **New_REST_000** object is created.

2. Rename the newly created connection as appropriate. The **New_REST_000 Parameters** view is displayed.

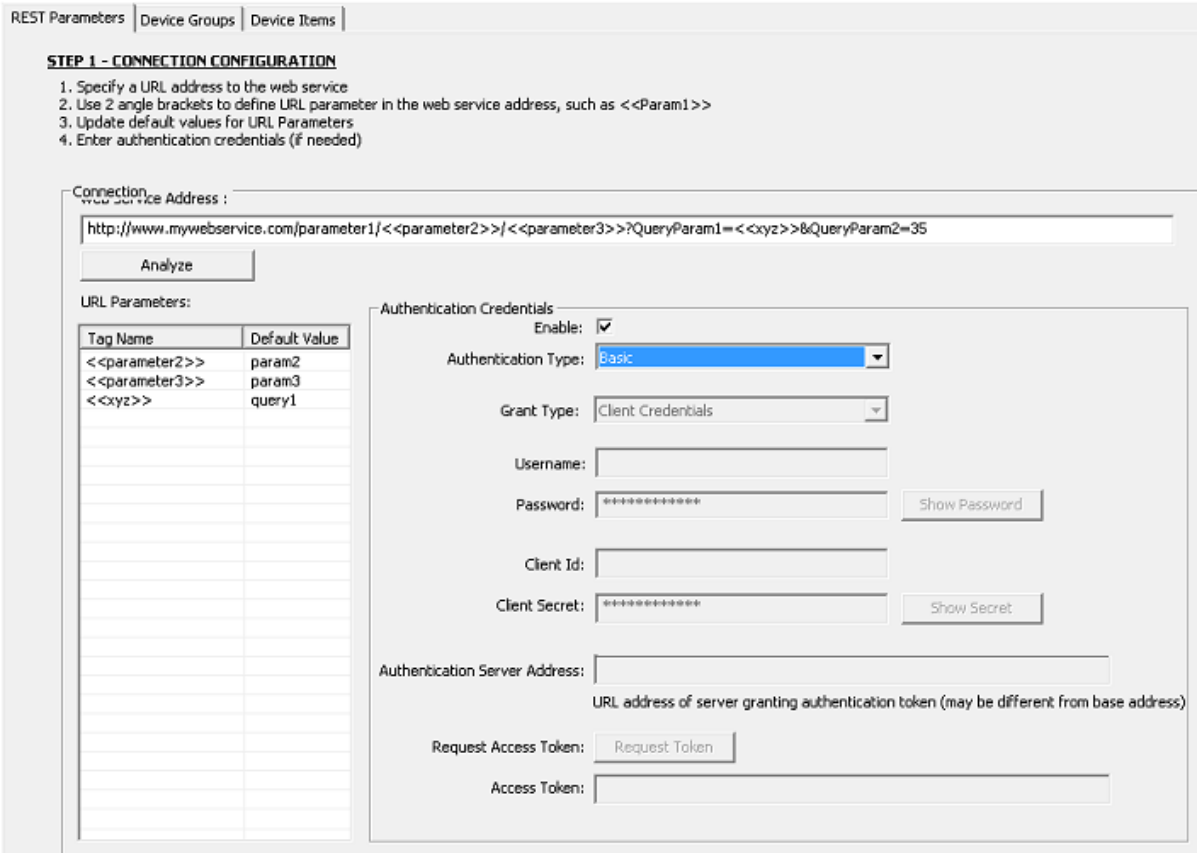
Configuring a REST Connection

Follow the steps to configure the REST Connection.

1. Connection Configuration
2. Interaction Method
3. Test and Map Preferences

Connection Configuration

This section allows you to analyze and authenticate the web service.



Analyzing the Web Service Address

1. In the **Web Service Address** field, enter the URL of the web service you want to connect. The syntax of the address is:

   ```
   http://www.mywebservice.com/parameter1/<<parameter2>>/<<parameter3>>?
   QueryParam1=abc&QueryParam2=xyz
   ```

   where;

   - "abc" and "xyz" are query parameters
   - Parameters 1, 2, and 3 are the resource parameters

     The web service address includes the URL of the web service, path parameters as required by the web service, and the query parameters (optional).

     Note: You can connect to a web service with an encrypted (https) or unencrypted (http) URL.

     The rules for a valid web service address are:

   - The **Web Service Address** can only contain a maximum of 255 characters.
   - You can use variables within the web address (optional). To convert any parameter to a variable, add the parameter within double angle brackets (<<..>>). See Using Parameters in Web Service Address.

     Note: During the text execution, each parameter becomes a tag that can be exported to the Device Items list.

2. Click **Analyze.**

The parameters (specified within the double angle brackets << >>) are extracted and populated in the **Tag Name** column of the **URL Parameters** grid. Each extracted parameter is set as a reference that can be addressed individually. For more information, See Using Parameters in Web Service Address.

3. To modify the default value of the parameters, double-click the **Default Value** column for the respective tag name.

Authenticating the Web Service

In the **Authentication Credentials** section, select the **Enable** check box to enable **Authentication Type**, and additional required fields.

1. For Basic Authentication:
   a. From the **Authentication Type** list, select **Basic**.
   b. Enter the username and password in the respective fields.

      Click **Show Password** to view the password you entered.
2. For OAuth Authentication:
   a. From the **Authentication Type** list, select **OAuth 2.0**
   b. This enables the **Grant Type** drop-down box.
      - If you select **Password Credentials**, enter the Username and Password in the **Username** and **Password** field respectively. If the client ID and client secret are required by the web service, enter the details in the **Client Id** and **Client Secret** fields.
      - If you select **Client Credentials**, the **Username** and **Password** field gets disabled. Enter the client ID and client secret details, in the **Client Id** and **Client Secret** fields respectively.
   c. If the token bearer string is provided by the web service, enter it in the **Access Token** field. You may ignore the other fields.
   d. If the token bearer string is not provided by the web service, enter the **Authentication Server Address**, **User Name**, and **Password**. If the client ID and client secret are required by the web service, enter the details in the **Client ID** and **Client Secret** fields.

      This enables the **Request Token** button.
   e. Click **Request Token**.
      - The token bearer number populates in the **Access Token** field.

Note: Based on the authentication mode, the connection strings (web service address) may vary from the Authentication Server URL. For OAuth authentication, it is possible that the "Authentication Server Address" URL is a different URL than the one you actually use in the "Web Service Address" URL to interact with the web service. This is because the authentication is performed through one URL that assigns the bearer token, and the data exchange is performed in a different URL that allows the data exchange.

---

1. Custom Authentication:
   a. From the **Authentication Type** list, select **Custom**.
   b. To enter the Key, double-click the **Key** column in the Header, Body and Authentication grids.
   c. To enter the value of the key, double-click the **Value** column of the respective Key row.

d. If the token bearer string is not provided by the web service, enter the **Authentication Server Address**. This enables the **Request Token** button.

e. Click **Request Token**.

- The token bearer number populates in the **Access Token** field.

f. If the token bearer string is provided by the web service, enter it in the **Access Token** field. You may ignore the other fields.

Using Parameters in Web Service Address

Parameters as Variables

Using variables in the address field allows the same basic web service connection to query a different result set, based on the variables used in the web service address itself. The variables can be tied to tag names to be set externally from an application. To convert any parameter to a variable, add the parameter within double angle brackets (<<..>>).

Example - Connecting to the Open Weather Map Web Service

To request the weather forecast for a specific city, for example, London, enter the the following URL in the **Web Service Address** field.

```
http://api.openweathermap.org/data/2.5/weather?q=London&appid={Your API Key}
```

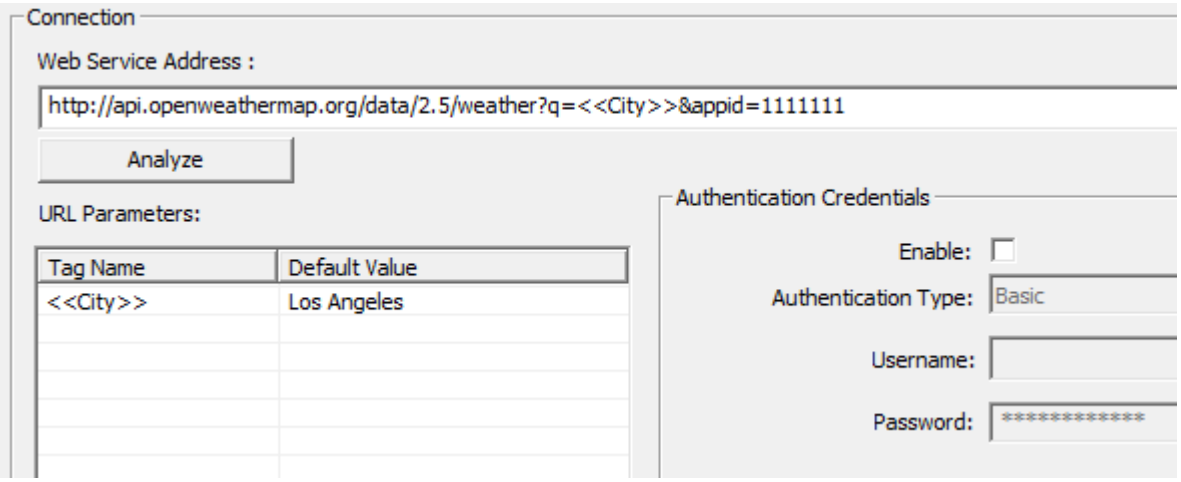
Note: The OpenWeather API requires a valid API Key to be submitted with the **appid** URL parameter as an identification of the caller. The API keys linked to an account are used to take count of the calls made to the OpenWeather platform, and to enforce the limit of free API calls permitted for the account. Visit the OpenWeather website to acquire your own API key for use with this example.

To convert the address in order to query any city, replace the city name 'London' with a variable - <<City>>. The web service address changes to:

```
http://api.openweathermap.org/data/2.5/weather?q=<<City>>&appid={Your API Key}
```

Upon analyzing the web service address, the **URL Parameters** grid populates the **Tag Name** column with the <<City>> variable.

To get the weather forecast of any city of your choice, set the **Default Value** of <<City>> variable to the city name and execute the test operation. URL Parameters are exposed as tags which can be set by consuming applications. For instance, the <<City>> parameter is accessible via the item name **opww.get.query.<<City>>**.



Parameters as References

When a web service address is analyzed, the WEBSVC Communication Driver exposes each parameter from the result set as references that can be easily addressed individually.

Example - Connecting to the Online Web Service

To connect to the Online web service, enter the online web service query in the Web Service Address field.

Note: You must have an account with AVEVA Insight to complete these steps. If you do not have a AVEVA Insight account, go to the AVEVA Insight site () and click **Sign Up**.

Using Basic Authentication

1. Enter the web service address: https://online.wonderware.com/s/<<Solution_ID>>/apis/Historian/v2/Tags.

   Each account has a unique Solution_ID.The Solution_ID entered within double angle brackets '<<..>>' is displayed as a parameter in the **URL Parameters** grid. To use the Solution_ID parameter as a variable (for instance, tecpny), set the parameter default value to tecpny, in the **URL Parameters** grid.

   Note: When using basic authentication, the web service address must include the Solution_ID as part of the URL.

2. Click **Analyze**.
3. In the **Authentication Credentials** section, select the **Enable** check box.
4. From the **Authentication Type** list, select **Basic**.
5. Enter the credentials used for the AVEVA Insight account in the **Username** and **Password** fields.
6. Click **Connect**.

Using OAuth Authentication

1. Enter the web service address: https://online.wonderware.com/apis/Historian/v2/Tags.
2. Click **Analyze.**
3. In the **Authentication Credentials** section, select the **Enable** check box.
4. From the **Authentication Type** list, select **OAuth 2.0**.
5. Enter the bearer token for the web service. The bearer token can be obtained from the **API Key Authentication** section of the AVEVA Insight website.
6. Click **Connect**.

Note: The OAuth 2.0 authentication uses a URL different from the web service URL.

Using Custom Authentication

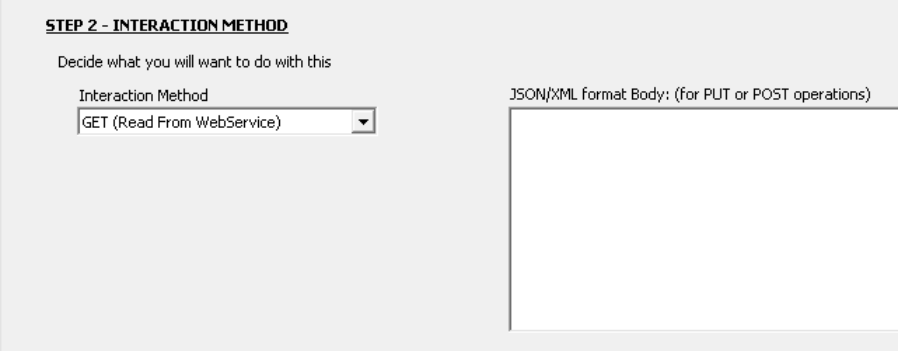
1. Enter the web service address: https://online.wonderware.com/apis/Historian/v2/Tags.
2. Click **Analyze.**
3. In the **Authentication Credentials** section, select the **Enable** check box.
4. From the **Authentication Type** list, select **Custom**.
5. Edit the existing values of the Header, Body and Authentication grids.
   - In the **Key** column of the Body grid, enter the AppId and Secret in different rows.

- In the **Value** column, enter the values for the AppId and Secret.

6. Enter the Authentication Server Address, and click **Request Token**.
7. The token populates in the Access Token field.

Interaction Method

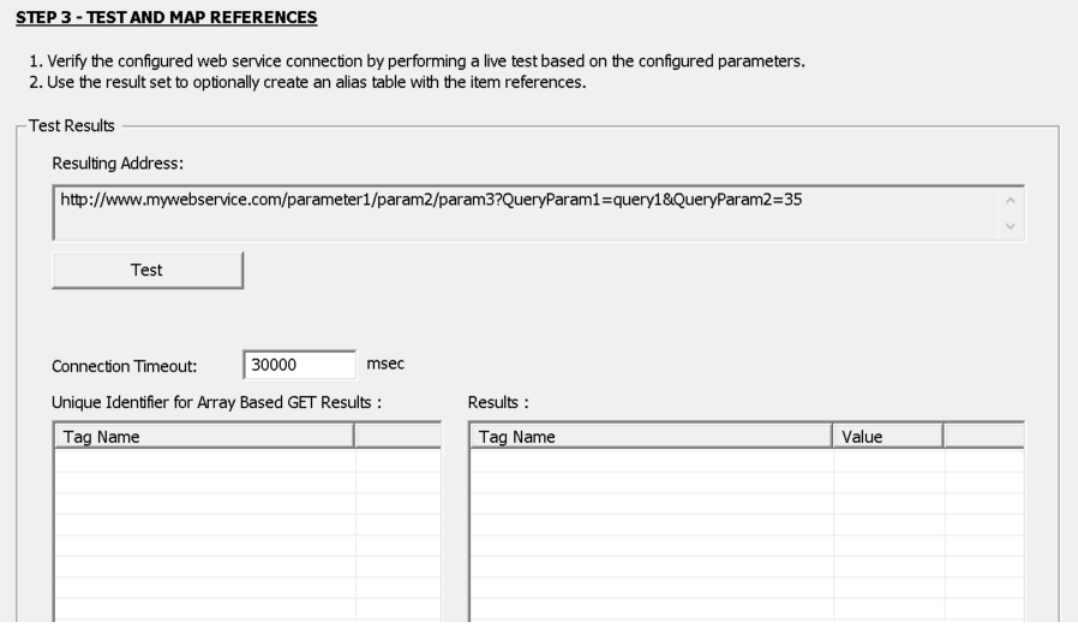
This section allows you to select the method to interact with the web service.



1. From **Interaction Method** drop-down list, select a method to interact with the web service.
   - GET (Read resource(s) from a Web Service)
   - POST (Create resource(s) in a Web Service)
   - PUT (Update resource(s) in Web Service)
   - DELETE (Delete resource(s) from Web Service)
2. If you select POST or PUT as the interaction method, enter the JSON or XML string as required by the web service in the **JSON/XML format Body** field.

Test and Map Preferences

This section allows you to verify the configured web service connection, and use the result set to create the table of item references.



Testing Web Service

To verify the configured web service, execute the test operation on the web service based on the configured paramters.

1. The **Resulting Address** section shows the resulting web URL.
2. Click **Test.**

   This allows the user to test the connection to the web service and also examine the results received. The results are displayed in the **Results** grid**.**

Note: The **Connection Timeout** is the time period for which this driver waits for connection to get established. The default value is 30000 milliseconds. You can change this field, if the connection fails.

Upon completion of test execution, the **Test Response Time** displays the time taken (in ms) by the web service to respond and the item count. Use the result to configure the minimum update interval of a topic. For more information, see Device Group Definitions.

Retrieving the Results

Upon completion of test execution, the **Results** grid is populated. The **Results** grid displays two columns - **Tag Name** and **Value**.

- **Tag Name**: represents the name of the query parameter.
- **Value**: displays the status value of the corresponding query parameter at runtime.

The resulting tags and values are not editable. The list of tags in the **Tag Name** column are the items available for subscription in run time. Each tag name is the equivalent of a PLC register. These are parsed by the Communication Drivers and the individual parameters are exposed as references.The application can map a reference to the times listed in the tagname column fields.

Note: To trigger the web service on demand, use the .run tag from the **Tag Name** column in the **Results** grid. Alternatively, you can poll using the device group update interval. For more information, see Triggering a Web Request.

Result Code of a Web Service Connection

The Result code is a value of the Status tag in the result tag list of the GET/PUT request executed.

- Result code = 0; no request started
- Result code = 1; request started
- Result code = 2; request successful
- Result code = 3; request failed

Note: The tags `$op$ww.errorcode` and `$op$ww.errormsg` are returned with every result set. For more information, see WEBSVC Communication Driver Error Codes.

Exporting to Device Items

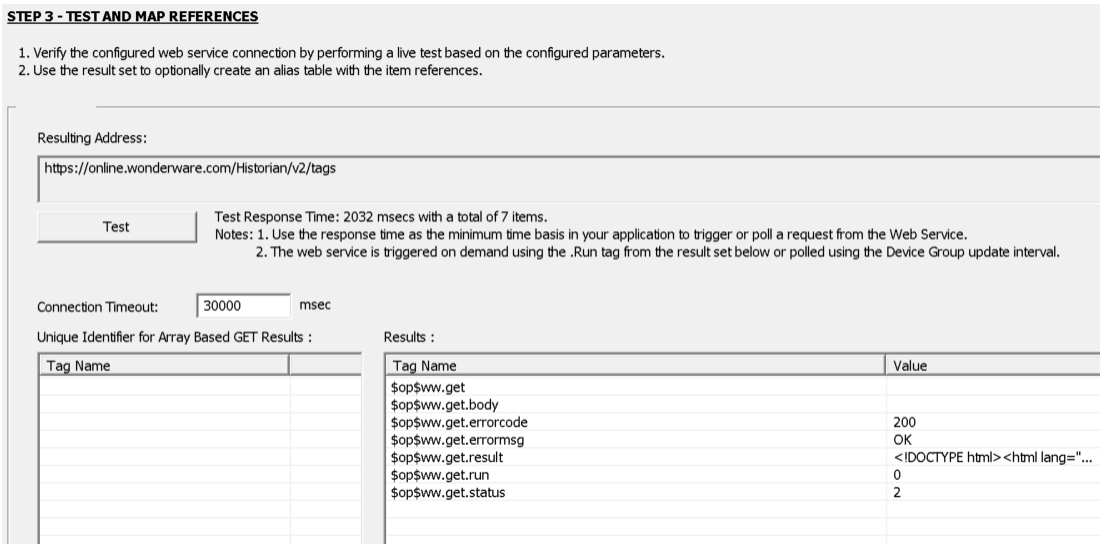
To export the result tags to the Device Items tab

- In the **Test and Map Preferences** section, click **Export to Device Items**.

  This allows the individual parameters from the result set reference table to be copied or exported to the **Device Items** table where a user-friendly alias name can be used if desired.

Determining a Unique Identifier

The REST-based web service connections return complex data sets in an array-based numbered structure format. Any modification to the array list in the web service can change the index, resulting in the change of the tag position.

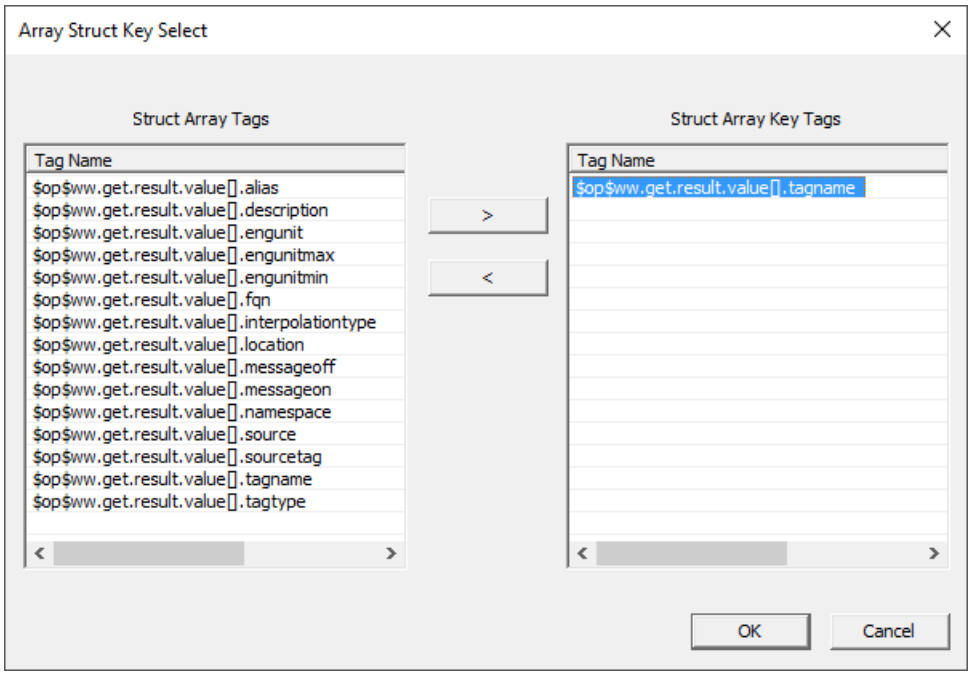


The WEBSVC Communication Driver allows array-based lists to be mapped to unique keys (unique identifier) such that the result set is exposed as a structure based on the unique name. To reference the result set by their name instead of by element numbers, you can map the numbered array elements to unique named elements within the result set.

You can select one or more elements of the tag name with unique value across the array results, and set as a unique key (identifier). When a test is executed again, value of the result set, which is now a name-based list instead of numeric index numbers, is displayed in all the tags.

1. Click **Select Struct Array Key…**

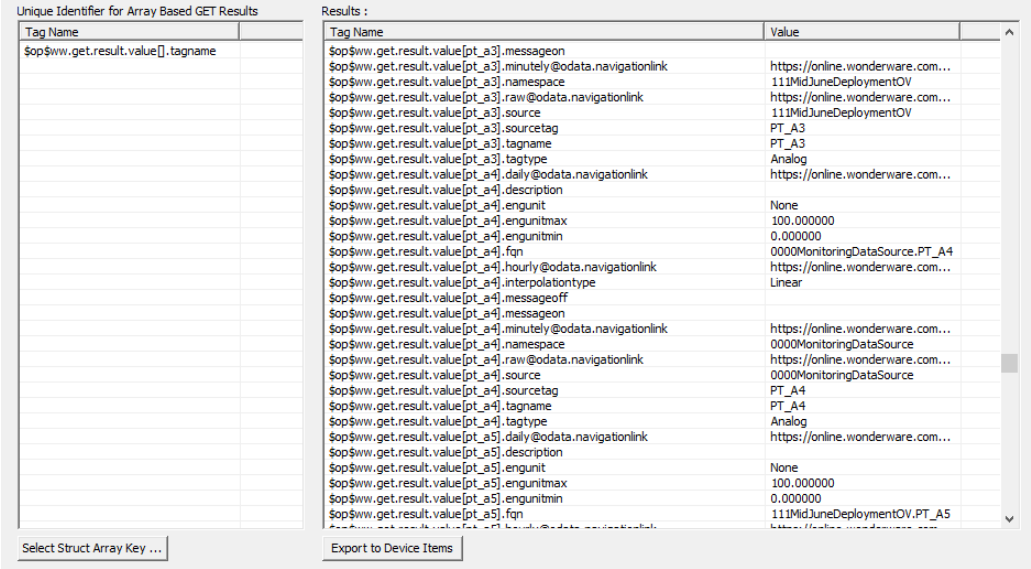
The **Array Struct Key Select** dialog appears.



2. From the **Struct Array Tags** (left pane), select the the tags to create a unique key index.
3. Click **>** to move the tag to the **Struct Array Key Tags** (right pane). Click **<** to remove the selected tag from the **Struct Array Key Tags** list.
4. Click **OK.**

   The selected array displays in the **Unique Identifier for Array based results** grid.
5. Click **Test** again.

   The unique key is inserted in the tag name results under **Unique Identifier for Array Based GET Results**.



Note: You can select both structured array tags and structured nested array tags as unique identifiers.

Device Group Definitions

The **Device Groups** configuration view is used to add, clear, rename, delete, import, and export device groups. It has the following two columns:

- **Name**: alias name to the actual data source items.
- **Update Interval (ms)**: frequency, in milliseconds, that the WEBSVC Communication Driver acquires data from the topics associated with that device group.

Different topics can be polled at different rates from a PLC by defining multiple device group names for the same PLC and setting a different update interval for each device group. For more information about automatic (time-based) and trigger-based (on-demand) polling, see Triggering a Web Request.

Note: Enter a unique name for each new device group added/created.

To create or add device groups

1. Right-click anywhere in the **Device Groups** configuration view, and select **Add**.

   A device group is created, and it is numerically named by default. For example, Topic_0, Topic _1, and so on.

2. To change the default name, double-click on it and enter the new name.

To edit the update interval

Double-click on the **Update Interval** entry and update with the new value.

Or

Right-click the **Update Interval** entry and click **Modify Update Interval**.

An update interval default to 10,000 is automatically added when a device group is created. For trigger-based polling, set the **Update Interval** to zero.

Note: The WEBSVC Communication Driver communicates with the target web service in two ways - Automatic and Trigger-based polling.For details about supported polling types, see Triggering a Web Request.

To rename a device group

1. Right-click on the device group to be renamed, and select **Rename.**
2. Enter the new device item name.
3. To apply the change, press **Enter** or click anywhere in the configuration view**.**

To delete a device group

- Right-click on the device group to be deleted and select **Delete**.

The device group and its corresponding update interval are deleted from the configuration view.

Note: When you select another part of the WEBSVC Communication Driver tree hierarchy, you are prompted to save the modifications to the configuration set.

Device Item Definitions

The Device Items configuration view is used to add, clear all, rename, delete, import, and export device items. It has the following two columns:

- **Name**: Defines the alias names to actual data source items.
- **Item Reference**: Defines the actual data source item names.

Note: Enter a unique name for each new device group added/created.

To create or add device items

1. Right-click anywhere in the **Device Items** configuration view, and select **Add.**

   A device item is created, and it is numerically named by default. For example, Item_0, Item_1, and so on.

2. To change the default name, double-click and enter the new name.

   Enter a unique name for the new device item.

To export the result tags to the Device Items

- In the Results section of either SOAP/REST configuration view, click **Export to Device Items**.

  This allows the individual parameters from the result set to be copied or exported to the **Device Items** table where a user-friendly alias name can be used if desired.

To add item references

1. In the **Item Reference** column, double-click the same horizontal line as the selected device item.
2. Type in the actual data source item name.
3. To apply the change, press **Enter** or click anywhere in the configuration view.

To rename a device item

1. Right-click on the device item, and select **Rename.**
2. Enter the new device item name.
3. To apply the change, press **Enter** or click anywhere in the configuration view.

To delete a device item

- Right-click the device item to be deleted, and select **Delete**.

  The device item and its corresponding data source item name are deleted from the configuration view.

To clear all device items

- Right-click anywhere in the **Device Items** configuration view, and select **Clear All**.

  All the device items listed in the configuration view, including their corresponding data source item names are deleted.

Note: When you select another part of the Communication Driver tree hierarchy, you are prompted to save the modifications to the configuration set.

Item Syntax for VTQ Timestamp

Each payload from a web service has an associated time value, that shows the time of occurrence of an event. By default, the Communication Driver time-stamps the references with the local time from the computer running the Communication Driver, and not the time the event occurred.To obtain an accurate timestamp value of a data field, the REST-based connections support the &T& syntax, that concatenates the data field and the time field.

<Value_Reference>&T&<Time_Reference>

If a value item is X and the time item is Y, the resulting read-only item X&T&Y displays the value X timestamped with the value of Y.

Prerequisites for using the &T& Item Syntax

- A time field must be available in the data record.
- The time field can be either:
  - Unix timestamp

    Example 1: 1507103430 = Wednesday October 4 2017 07:50:30 UTC

    Example 2: 1507103432 = Wednesday October 4 2017 07:50:32 UTC
  - ISO 8601 timestamp string in UTC format

    Example 1: 2017-10-04T07:50:30Z = Wednesday October 4 2017 07:50:30 UTC)

    Example 2 (when time is in ms): 2017-10-04T07:50:30.134Z = Wednesday October 4 2017 07:50:30.134 UTC).

    Note: The ISO 8601 timestamp string ends with Z.

Example

Consider the following payload, where the item reference is Tank_Temp, and time stamp is available in both ISO and Unix formats.

Tank_Temp {"Temp":11.6, "TimeISO":" 2017-09-07T10:05:02.000Z", "TimeUnix": 1505392739}

- To get the temperature time stamped with ISO Time:

  Tank_Temp.Temp&T&Tank_Temp.TimeISO

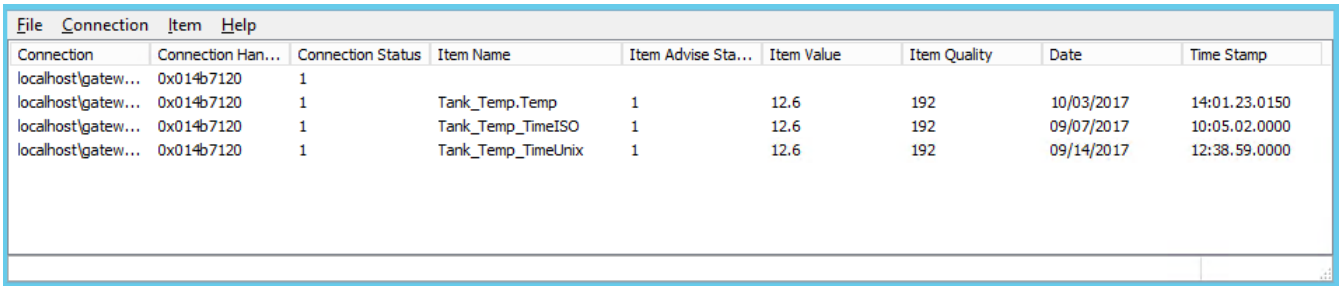
  or

  Tank_Temp.Temp&T&.TimeISO
- To get the temperature time stamped with Unix Time:

  Tank_Temp.Temp&T&Tank_Temp.TimeUnix

  or

  Tank_Temp.Temp&T&.TimeUnix



Hot Configuring the WEBSVC Communication Driver

The changes to the hot configurable parameters take effect while the Communication Driver is running. The hot configurable parameters in WEBSVC Communication Driver are:

- Modifying global configuration parameters, device groups, and device items
- Adding a device node
- Modifying a device node
- Deleting a device node

Note: The server-specific configuration parameters are not hot configurable. Changes take effect only after the Communication Driver is restarted/reset.

AVEVA

Chapter 4

Accessing Data using the WEBSVC Communication Driver

Tag Naming Conventions

The Tag Name syntax for SOAP and REST connections is

```
<Instance>.<Operation>.<Support>.<Tag>
```

where:

- **Instance**: unique object within the Communication Driver
- **Operation**: operation to be executed
- **Support**: status of the operation
- **Tag**: name of the tag

Instance Naming Rules

- The instance name must be less than 20 characters long.
- The instance name must be unique within each node of the SOAP or REST hierarchy.
- The instance name must be unique for each specified operation within the same node.

Support Names and Description

There are five support name options.

- **run** - initiates execution of an operation.
- **inputs/query** - contains one or more inputs to be set before executing an operation.

  Note: The term 'input' is used for SOAP, and 'query' is used for REST.
- **status** - indicates the progress, success, or fail of an executed operation.

  (1=Executing, 2=Success, 3=Failed)

- **result** - contains all the result tags generated from an operation. For more information on the errorcode and error messages, see Troubleshooting the WEBSVC Communication Driver.
- **errormsg** - indicates an error message when an operation fails to execute.

Tag Naming Rules

- The tag name can have many additional levels, each separated by "."
- The number of levels for 'result' is determined by the response data received from the WEBSVC operation execution.

Example

Consider two separate instances - **Phone1** and **Phone2** with the same operation name: **CheckPhoneNumber**. Phone1 is used to execute the CheckPhoneNumber operation from Client 1. Phone2 is used to execute the CheckPhoneNumber operation from Client 2.

Instance Phone1

opphone1.checkphonenumber.inputs.licensekey

opphone1.checkphonenumber.inputs.phonenumber

opphone1.checkphonenumber

opphone1.checkphonenumber.errormsg

opphone1.checkphonenumber.result

Instance Phone2

opphone2.checkphonenumber.inputs.licensekey

opphone2.checkphonenumber.inputs.phonenumber

opphone2.checkphonenumber

opphone2.checkphonenumber.errormsg

opphone2.checkphonenumber.result

Triggering a Web Request

The WEBSVC Communication Driver supports both automatic and trigger-based polling

Automatic (Time-based) Polling

The WEBSVC Communication Driver supports automatic polling to trigger a data request, where a polling period can be configured. Different topics can be polled at different rates from a PLC by defining multiple device group names for the same PLC and setting a different update interval for each device group. Set the Update interval polling period to a number (in milli-seconds). The WEBSVC Communication Driver automatically requests the data from the web service on the period cycle.

The polling period should be set greater than the test response time. This ensures that the runtime collects data from the driver after a trigger.

When a test is performed, the reply test duration is displayed The reply test duration indicates the time taken to complete the requested operation. The **Update Interval** of a topic can be tuned based on the reply test duration.

Trigger-based (On Demand) Polling

The trigger-based (on demand) polling allows the web service to trigger the request on command. To perform trigger-based polling, set the update interval time to zero to prevent automatic polling. To force the web service to perform an update, set the "opww.get.run" reference to **True**. Upon the execution of the operation, the web service will reset the parameter to False.

If a configuration change, such as a configuring the topic update interval, is made when the WEBSVC Communication Driver is running, it is highly recommended to perform a reset on the corresponding hierarchy.

- **Trigger-Based Polling for SOAP Connection:** Triggering or Auto-Poll can be applied to any SOAP method, because it cannot be determined if the method is writing or reading from the web service.
- **Trigger-Based Polling for REST Connection:** Triggering is required for POST, PUT, and DELETE. Triggering or Auto-Poll can be applied to GET.

Note: It is not recommended to mix topics with Trigger-based polling (zero update interval) and the topics with Time-based polling (non-zero update interval). If using Trigger-based polling, set all topics to zero.

Exporting and Importing Data Items

After configuring the device items, you can export and import the Communication Driver item data to and from a CSV file. This will allow you to perform an off-line, large-scale edit on the item data configured for a controller, and import what has been edited back into the controller configuration.

To export Communication Driver item data to a CSV file

1. Right-click anywhere in the **Device Items** configuration view.
2. Select the **Export** command from the shortcut menu.
   - The **Save As** dialog box appears.
   - The file name defaults to NEW_SOAP_000.csv or NEW_REST_000.csv based on the current node.
3. Accept the defaults to save the file or rename the file if appropriate.

The file can now be edited off-line in Microsoft Excel. It contains one row for each item configured with two columns - **Name** and **Item Reference**.

To import Communication Driver item data from a CSV file

1. Right-click in the **Device Items** configuration view.
2. Select the **Import** command from the shortcut menu.
   - The **Open** dialog box appears.
   - It defaults to the .csv file extension within the current-system-configured default directory.
3. Browse and select the specific CSV file you want to import, and click **OK** for confirmation.

   The OI Server Manager will import the file and deposit it in the **Device Items** box.

During the imported file processing:

- New item references are added based on unique names.
- If there are duplicate names, you can either replace the existing entry with the new entry, or ignore the new entry.

When the Communication Driver is running and an OPC client requests item information, the imported configured items will show up under the controller hierarchy node.

Chapter 5

Troubleshooting the WEBSVC Communication Driver

Using the Diagnostic Node

The WEBSVC Communication Driver supports same diagnostic system items as other AVEVA Communication Drivers, along with the following protocol specific system items.

Name	Type	Description
sysdiagnosticlogging	VT_UI4	When poked, turns on runtime logging. (Read Only) NONE = 0 INFO = 1 WARNING = 2 ERROR = 4 TRACE = 8 **Note:** Flag values can be added together to enable multiple at the same time. i.e. 1+8 = 9 which enables INFO and TRACE.

For more information, see the AVEVA Communication Drivers Pack Help.

Using the WEBSVC Communication Driver Diagnostic Log

The WEBSVC Communication Driver provides additional logging capability. This capability can be turned on using a system tag named sysdiagnosticlogging.

Diagnostic logging records the previous three hours. The log lines are output to a file on disk in the WEBSVC Communication Driver data folder:

C:\ProgramData\Wonderware\OI-Server\$Operations Integration Supervisory Servers$\OI.WEBSVC\LogFiles

WEBSVC Communication Driver Error Codes

The following two tags are returned with every result set.

1. opww.errorcode
2. opww.errormsg

opww.errorcode

The tag opww.errorcode is the numeric representation of the result code return when the web service query is executed.

- Errorcode within the range - 200 to 299 is considered to be a success.
- Any error code outside of the 200-299 range is considered to be a failure.
- The error code 900 indicates a failure locally on the WEBSVC Communication Driver side of the connection. The error message shows more detailed information about the 900 error.

For more information of HTTP error codes that a web service can return, see .

opww.errormsg

The tag opww.errormsg is the textual interpretation of the errorcode received. If the query result is “OK”, it means the query result was successful. The ErrorMsg tag values are returned from the web service and is determined by the web service itself.