TIDAL

Tidal Workload Automation Web Services SOAP API Guide

Version 6.3.3

First Published: January 2018

tidalautomation.com

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS. THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE PRODUCTS IN THIS MANUAL ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR STA GROUP REPRESENTATIVE FOR A COPY.

The implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. © 1981 Regents of the University of California. All rights reserved.

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

All printed copies and duplicate soft copies are considered uncontrolled copies and the original online version should be referred to for latest version.

© 2018 STA Group LLC. All rights reserved.

Contents

Preface

This guide describes how to generate and use the Tidal Workload Automation Web Services SOAP API.

Audience

This guide is for engineers who want to integrate applications or systems with TWA for Workflow Management activities.

Related Documentation

For a list of all Tidal Workload Automation guides, see the *Tidal Workload Automation Documentation Overview* of your release on tidalautomation.com at:

http://docs.tidalautomation.com/

Note: We sometimes update the documentation after original publication. Therefore, you should also review the documentation on tidalautomation.com for any updates.

Obtaining Documentation and Submitting a Service Request

For information on obtaining documentation, submitting a service request, and gathering additional information, see What's New in Tidal Product Documentation at:

https://docs.tidalautomation.com/rss

Subscribe to What's New in Tidal Product Documentation, which lists all new and revised Tidal technical documentation, as an RSS feed and deliver content directly to your desktop using a reader application. The RSS feeds are a free service.

Document Change History

The table below provides the revision history for the *Tidal Workload Automation Web Services SOAP API Guide*.

Table 1

Version Number	Issue Date	Reason for Change
6.3	August 2016	Rebranded Cisco Tidal Enterprise Scheduler (TES) to Cisco Workload Automation (CWA).
6.3.3	January 2018	Rebranded Cisco Workload Automation (CWA) to Tidal workload Automation (TWA).

1

Introduction

This chapter provides an overview of the Tidal Workload Automation Web Services SOAP API.

Overview

This Application Programming Interface (API) documentation describes the Web services API used with Tidal Workload Automation (TWA). This documentation describes the XML formatting used to present the input and output of API calls processed via Web services.

The TWA Web Service SOAP API delivers programmatic access methods to TWA objects through a Web Service interface. The service is provided by the TWA Plug-in which resides on a Client Manager.

2
Web Services Procedures

This chapter describes how to generate the Tidal Workload Automation (TWA) Web Services SOAP API.

Overview

The TWA Web Services SOAP API is published through a Web Service Definition Language (WSDL) document and its associated schema. The WSDL file defines available Web Service operations, binding, and access end points. The schema file defines operation parameters and additional data types.

Setting Up the Client User

The TWA Web Services SOAP API leverages the same authentication mechanism as the Tidal Workload Automation Web UI. The login for the client program requires a combination of username and password known to the TWA . To implement authentication for your client, the easiest way is to utilize the standard BindingProvider API of JAX-WS as shown in the code snippet below. More details are available in the supplied sample client in <TESSoapClient>\src\com\tidalsoft\tesws\client\Console.java.

```
TESWebService_Service tesws = new TESWebService_Service();

TESWebService teswsPort = tesws.getTESWebServicePort();

BindingProvider bindingProvider = (BindingProvider)teswsPort;

Map<String, Object> map = bindingProvider.getRequestContext();

map.put(BindingProvider.USERNAME_PROPERTY, username);

map.put(BindingProvider.PASSWORD_PROPERTY, password);
```

Before carrying out each operation, the Web Services API validates the given user account against the security and object access policy that the user has defined in TWA to determine if the account has the privilege to perform that operation.

Generating the TWA Web Services SOAP API

This section describes how to generate the TWA Web Services SOAP API.

To generate the TWA Web Services SOAP API

1. In a live TWA environment, go to the following websites to download the WSDL file and its associated schema files:
   - http://<hostname>:<port>/api/<DSP Name>/webservice/TESWebService?WSDL
   - http://<hostname>:<port>/api/<DSP Name>/webservice/TESWebService_schema1.xsd
2. Enter the following substitutions:
   - **<hostname>** – Host name used to access the TWA Web Client (TWA Web UI).
   - **<port>** – Port number used to access the TWA Web Client (TWA Web UI).

- **<DSP Name>** – Name of Data Source Provider of the target TWA Plug-in, which is the name that the user specifies when installing a TWA Plug-in.

These documents conform to the following specifications:

- http://www.w3.org/TR/wsdl (SOAP 1.1)
- http://www.w3.org/TR/2002/WD-wsdl12-20020709/ (SOAP 1.2)

3. Generate the client source code in Java, C#, or other languages using a tool, such as **wsimport** that comes with JDK 1.6, as shown in the following example:

```
wsimport -keep -verbose -extension -s <Target Source Directory> -d <Target Build Directory> -p
<Target Package Name> <WSDL File>
```

4. Use the generated object classes to complete the client application.

For information on the sample client provided, see Sample Client, page 21.

For information on using the dateTime parameters in the TWA Web Services SOAP API, see Date/Time Parameters, page 23.

3
Operations

This chapter describes the operations in the TWA Web Services SOAP API.

Operations

This section provides an overview of the TWA Web Services SOAP API by describing the operations in the API. It is recommended that you review the WSDL and schema files for details of each operation and its parameters.

addrule

The **addrule** operation defines a new job rule (job definition). It can be followed by the **modrule** operation to further define other job parameters.

After **addrule** has been issued, use the **submit** operation to add the job to the production schedule.

```
<operation name="addrule">
      <input message="tns:addrule"/>
      <output message="tns:addruleResponse"/>
    </operation>
```

agent

The **agent** operation enables or disables a connection to an agent provided you have the correct security privileges for editing this connection.

```
<operation name="agent">
      <input message="tns:agent"/>
      <output message="tns:agentResponse"/>
    </operation>
```

alerts

The **alerts** operation displays all the operator alerts presently in the production schedule in table format. The columns included are:

- **ID** – The alert ID (needed for the **alertset** operation).
- **Job Number** – The job number ID of the job that issued the alert.
- **Type** – The kind of alert issued.
- **Level** – The severity level of the alert, either Critical, Warning, Error, or Information.

- **Status** – The status of the alert, either Open(1), Acknowledged(2), or Closed(3).
- **Description** – The alert message as defined in the operator alert action used to issue the alert.
- **Response** – Operator notes taken in response to the alert.
- **Time** – The time the operator alert was closed.
- **User** – The runtime user of the job that closed the alert.

```
<operation name="alerts">
<input message="tns:alerts"/>
<output message="tns:alertsResponse"/>
</operation>
```

alertset

The **alertset** operation lets you manually set the status of an alert specified by the alert ID. To obtain the job number alert ID, use the **alerts** operation.

```
<operation name="alertset">
      <input message="tns:alertset"/>
      <output message="tns:alertsetResponse"/>
    </operation>
```

calendars

The **calendars** operation displays a list of calendars accessible to the current user.

```
<operation name="calendars">
      <input message="tns:calendars"/>
      <output message="tns:calendarsResponse"/>
    </operation>
```

calrecalc

The **calrecalc** operation recalculates all calendar dates.

```
<operation name="calrecalc">
      <input message="tns:calrecalc"/>
      <output message="tns:calrecalcResponse"/>
    </operation>
```

compile

The **compile** operation compiles the production schedule for the dates specified.

```
<operation name="compile">
```

```
        <input message="tns:compile"/>
        <output message="tns:compileResponse"/>
    </operation>
```

delrule

The **delrule** operation deletes a job or job group definition. You can either specify the alias or the ID of the job or job group.

```
<operation name="delrule">
        <input message="tns:delrule"/>
        <output message="tns:delruleResponse"/>
    </operation>
```

depadd

The **depadd** operation adds a new job dependency or file dependency.

```
<operation name="depadd">
        <input message="tns:depadd"/>
        <output message="tns:depaddResponse"/>
    </operation>
```

depdel

The **depdel** operation deletes job dependencies and file dependencies. It then replaces that job's predone instances in the production schedule.

Refer to the **submit** operation.

```
<operation name="depdel">
        <input message="tns:depdel"/>
        <output message="tns:depdelResponse"/>
    </operation>
```

getmasterstatus

The **getmasterstatus** operation provides the operational status of the UNIX master being used.

```
<operation name="getmasterstatus">
        <input message="tns:getmasterstatus"/>
        <output message="tns:getmasterstatusResponse"/>
    </operation>
```

getmasterversion

The **getmasterversion** operation displays the version of the UNIX master being used.

```
<operation name="getmasterversion">
    <input message="tns:getmasterversion"/>
    <output message="tns:getmasterversionResponse"/>
</operation>
```

grpupd

The **grpupd** operation updates inherit attributes for jobs in the specified group. You can obtain the group's Job ID by using the **grpupd** operation.

```
<operation name="grpupd">
    <input message="tns:grpupd"/>
    <output message="tns:grpupdResponse"/>
</operation>
```

historyPurge

The **historypurge** operation will delete the job history that is stored in the database.

```
<operation name="historyPurge">
    <input message="tns:historyPurge"/>
    <output message="tns:historyPurgeResponse"/>
</operation>
```

hosts

The **hosts** operation lists information about all TWA hosts defined in TWA .

```
<operation name="hosts">
    <input message="tns:hosts"/>
    <output message="tns:hostsResponse"/>
</operation>
```

inactrule

The **inactrule** operation inactivates or disables a job or job group. When a job or job group is inactivated, its occurrences (if any) are pulled from the production schedule.

```
<operation name="inactrule">
    <input message="tns:inactrule"/>
    <output message="tns:inactruleResponse"/>
```

```
    </operation>
```

jobadd

The **jobadd** operation lets you add a job or job group to the production schedule. You can add a job either by alias or by ID. You can obtain the job occurrence ID and/or alias by using the **listrule** operation. Unlike the **submit** operation, the job is submitted adhoc. An adhoc job is not dependent on a calendar because a new instance is submitted manually into the schedule.

```
<operation name="jobadd">

      <input message="tns:jobadd"/>

      <output message="tns:jobaddResponse"/>

    </operation>
```

jobcancel

The **jobcancel** operation cancels a job occurrence with the specified job ID from the production schedule. You can also specify whether canceling the job affects other dependent jobs. You can obtain the ID by running the **jobmon** operation.

```
<operation name="jobcancel">

      <input message="tns:jobcancel"/>

      <output message="tns:jobcancelResponse"/>

    </operation>
```

jobdep

The **jobdep** operation displays all the dependencies of the specified type belonging to a job or job group.

```
<operation name="jobdep">

      <input message="tns:jobdep"/>

      <output message="tns:jobdepResponse"/>

    </operation>
```

jobgo

The **jobgo** operation overrides all dependencies for a job or job group, allowing the job or job group to launch. Obtain job run IDs by running the **jobmon** operation.

```
<operation name="jobgo">

      <input message="tns:jobgo"/>

      <output message="tns:jobgoResponse"/>

    </operation>
```

jobhold

The **jobhold** operation prevents a job occurrence with the specified job occurrence ID from running in the production schedule. Obtain the job occurrence ID by running the **jobmon** operation.

```
<operation name="jobhold">

      <input message="tns:jobhold"/>

      <output message="tns:jobholdResponse"/>

    </operation>
```

jobmod

The **jobmod** operation modifies a Job Occurrence. You can obtain the job run ID by using the **jobmon** operation.

```
<operation name="jobmod">

      <input message="tns:jobmod"/>

      <output message="tns:jobmodResponse"/>

    </operation>
```

jobmon

The **jobmon** operation enables you display job occurrence information. Command options allow you to filter the information displayed.

```
<operation name="jobmon">

      <input message="tns:jobmon"/>

      <output message="tns:jobmonResponse"/>

    </operation>
```

jobrelease

The **jobrelease** operation resumes a job occurrence that was held with the **jobhold** operation and releases a job in a **Waiting on Operator** status. You can obtain the job run ID by using the **jobmon** operation.

```
<operation name="jobrelease">

      <input message="tns:jobrelease"/>

      <output message="tns:jobreleaseResponse"/>

    </operation>
```

jobrermove

The **jobremove** operation removes job occurrences from the production schedule. The job must have a prelaunched status. Once a job reaches Launched, Active, or one of the Completed statuses, it cannot be removed from the schedule. You can obtain the job rule ID by running the **listrule** operation.

```
<operation name="jobremove">
```

```
        <input message="tns:jobremove"/>
        <output message="tns:jobremoveResponse"/>
    </operation>
```

jobrerun

The **jobrerun** operation lets you manually rerun a completed job or job group. You can obtain the job run ID by using the **jobmon** operation.

```
<operation name="jobrerun">
        <input message="tns:jobrerun"/>
        <output message="tns:jobrerunResponse"/>
    </operation>
```

jobset

The **jobset** operation lets you manually set the completion status of a job. You can obtain the job run ID by using the **jobmon** operation.

```
<operation name="jobset">
        <input message="tns:jobset"/>
        <output message="tns:jobsetResponse"/>
    </operation>
```

listrule

The **listrule** operation lists information about job and job group rules. You can choose the information to list. You can also select which records to display.

```
<operation name="listrule">
        <input message="tns:listrule"/>
        <output message="tns:listruleResponse"/>
    </operation>
```

liststat

The **liststat** operation lists all possible job statuses and their associated status number in a two column format.

```
<operation name="liststat">
        <input message="tns:liststat"/>
        <output message="tns:liststatResponse"/>
    </operation>
```

mastershutdown

The **mastershutdown** operation shuts down the UNIX master.

```
<operation name="masterShutDown">
      <input message="tns:masterShutDown"/>
      <output message="tns:masterShutDownResponse"/>
    </operation>
```

modrule

The **modrule** operation modifies a job or job group definition (rule). To modify the rule, either the job ID or an alias must be specified.

```
<operation name="modrule">
      <input message="tns:modrule"/>
      <output message="tns:modruleResponse"/>
    </operation>
```

output

The **output** operation displays the output of a job.

```
<operation name="output">
      <input message="tns:output"/>
      <output message="tns:outputResponse"/>
    </operation>
```

pause

The **pause** operation temporarily halts the production schedule thereby preventing jobs, even those whose dependencies have been met, from being launched. To restart the production schedule, use the **resume** operation.

```
</operation>
    <operation name="pause">
      <input message="tns:pause"/>
      <output message="tns:pauseResponse"/>
    </operation>
```

qlimit

The **qlimit** operation lets you manually set the limit of an existing queue.

```
<operation name="qlimit">
      <input message="tns:qlimit"/>
```

```
        <output message="tns:qlimitResponse"/>

      </operation>
```

resume

The **resume** operation resumes the production schedule, allowing jobs to be launched after using pause to temporarily stop the production schedule. This does not apply to jobs that are waiting on dependencies.

```
<operation name="resume">

      <input message="tns:resume"/>

      <output message="tns:resumeResponse"/>

    </operation>
```

status

The **status** operation displays the status of a job or job group instance.

Status is the condition or state of a job occurrence throughout its life cycle. When a job has entered the schedule and is waiting to run or is actively running, possible statuses include:

- Active
- Waiting on Children
- Launched
- Waiting on Resource
- Waiting On Dependencies
- Held
- Agent Unavailable
- Agent Disabled
- Waiting on Group
- Timed Out for Day
- Waiting on Operator

When a job completes, possible status values are:

- Completed Normally
- Completed Abnormally
- Error Occurred
- Orphaned
- Skipped
- Aborted

- Cancelled
- Timed Out

If the status of a job occurrence is **Externally Defined**, then TWA is waiting for an external status update.

```
<operation name="status">
      <input message="tns:status"/>
      <output message="tns:statusResponse"/>
    </operation>
```

submit

The **submit** operation replaces predone job or job group instances in the production schedule according to its calendar. If there are job or job group instances already completed, these instances are not replaced.

```
<operation name="submit">
      <input message="tns:submit"/>
      <output message="tns:submitResponse"/>
    </operation>
```

varset

The **varset** operation lets you manually set the value of an already existing variable.

```
<operation name="varset">
      <input message="tns:varset"/>
      <output message="tns:varsetResponse"/>
    </operation>
```

4

Sample Client

This chapter describes the file structure of the sample client package and how to launch it.

Sample Client

The sample client is prebuilt using JDK 1.6. JRE v1.6 or above is required to run the sample client.

The TESSoapClient.zip file is located on CD2: WebServiceSample\JavaSoapClient.

The TWA Soap Client package contains complete source and binary files in the following folders:

- **bin** – Hand-coded client application startup scripts
- **config** – Hand-coded client application configuration scripts
- **docs**
  - **clientapi** – Javadoc generated from the client API package **com.tidalsoft.tesws.clientapi** in the src folder
  - **wsdl** – WSDL and schema file
- **lib** – .jar file compiled from the sources in the src folder
- **src**
  - **com\tidalsoft\tesws\clientapi** – Client code generated from the WSDL
  - **com\tidalsoft\tesws\client** – Hand-coded client application logic

Note: The WSDL files and the Javadoc in the docs folder provide an understanding of the WSDL and generated client code.

To launch the sample client

1. In a text editor, open **<TESSoapClient>\config\tescmd.props**.
2. Find **JAVA_HOME-TBD** and replace it with the fully qualified path to the JRE installation on the system.
3. Save the file.
4. Complete one of the following steps with the following substitutions:
   - **<TESSoapClient>** – Root directory into which the sample client is unzipped
   - **<endpointurl>** – Endpoint URL of the TWA Web Service to which the client will connect, such as http://<hostname>:<port>/api/<DSP Name>/webservice/TESWebService
   - **<username>** and **<password>** – User account used to log into the Client Manager

   a. For Windows, open a **Command Prompt** window and run the following:

   <TESSoapClient>\bin\sampleClient.cmd -endpointurl <endpointurl> -user <username> -pass <password>

   b. For Unix or Linux, open a shell command window and run the following:

Sample Client

Make sure that **executable** bit is turned on for the sampleClient.sh file.

<TESSoapClient>/bin/sampleClient.sh -endpointurl <endpointurl> -user <username> -pass <password>

5. To view commands, type **help** at the SACmd prompt in the command line interface.

5

Date/Time Parameters

This chapter describes how to use the date/time parameters in the Tidal Workload Automation Web Services SOAP API.

Date/Time Parameters

This section provides the following guidelines for using the date/time parameters in the TWA Web Services SOAP API:

- All date/time parameters are of XML **dateTime** type as defined in http://www.w3.org/TR/xmlschema-2/#dateTime.
- All date/time parameters are displayed and must be entered in the format conforming to this specification, with a TWA specific requirement that all times are according to the time zone of the target TWA Master.
- When entering the date/time parameter, it is recommended to leave the Time Zone field blank so that the time will be interpreted by the API according to the time zone of TWA Master.

  If present, the time zone field must represent the time zone of the target TWA Master. A typical date time string looks like one of the following:

  - With the time zone: 2010-05-03T09:19:03-05:00
  - Without the time zone: 2010-05-03T09:19:03
- The src/com/tidalsoft/tesws/client/util/DateTime.java source file in the sample client implements a pair of examples on creating proper XMLGregorianCalendar objects to represent date and time parameters.