Remote Manager

Programmer Guide

Revision history—90001437-13

Version	Date	Description
H	May 2016	Remote Manager platform release that includes the following features and enhancements: ■ Added expiration timeouts for health metric data. See Setting health metrics preferences. ■ Performance improvements and miscellaneous editorial corrections.
I	September 2016	Remote Manager platform release that includes the following features and enhancements: ■ Added ability to link Remote Manager accounts together such that a parent account can access and manage subaccounts. See About subaccounts. ■ Performance improvements and miscellaneous editorial corrections.
J	February 2017	Remote Manager platform release that includes the following new features and enhancements: ■ Added support for secure provisioning. See Adding and removing devices for information on adding devices that require an installation code. ■ Performance improvements and miscellaneous editorial corrections.
K	May 2018	■ Added a Get Started section to the documentation. ■ Edited documentation.

Trademarks and copyright

Digi, Digi International, and the Digi logo are trademarks or registered trademarks in the United States and other countries worldwide. All other trademarks mentioned in this document are the property of their respective owners.

© 2019 Digi International Inc. All rights reserved.

Disclaimers

Information in this document is subject to change without notice and does not represent a commitment on the part of Digi International. Digi provides this document “as is,” without warranty of any kind, expressed or implied, including, but not limited to, the implied warranties of fitness or merchantability for a particular purpose. Digi may make improvements and/or changes in this manual or in the product(s) and/or the program(s) described in this manual at any time.

Warranty

To view product warranty information, go to the following website:

www.digi.com/howtobuy/terms

Customer support

Gather support information: Before contacting Digi technical support for help, gather the following information:

- Product name and model
- Product serial number (s)
- Firmware version
- Operating system/browser (if applicable)
- Logs (from time of reported issue)
- Trace (if possible)
- Description of issue
- Steps to reproduce

Contact Digi technical support: Digi offers multiple technical support plans and service packages. Contact us at +1 952.912.3444 or visit us at www.digi.com/support.

Feedback

To provide feedback on this document, email your comments to

techcomm@digi.com

Include the document title and part number (Digi Remote Manager Programmer Guide, 90001437-13 K) in the subject line of your email.

Contents

Get started

Remote Manager concepts

Web services HTTP client applications

Web services conventions and versioning

Web services reference

Deprecated APIs

DIA (device integration application)

ISO 8601 date and duration reference

HTTP interface specification

UI descriptor reference

Get started

About Remote Manager programming

Remote Manager is a machine-to-machine (M2M) cloud-based network operating platform that includes a variety of Application Programming Interfaces (API's). Remote Manager supports:

- Application to device data interaction (messaging)
- Application and device data storage
- Remote management of devices

Devices are associated with the server through the Internet or other wide area network connection, which allows for communication between the device, server, and your applications. An important part of this communication is the transfer of data from a device to the server. Users can control devices by sending commands to devices or scheduling tasks. Devices can upload data to Data Streams on Remote Manager which are then available for push notifications, alarms, or retrieval by Web Services clients. Device communication can be achieved using Remote Manager devices, DIA (Device Integration Application), or connecting your own device using Digi Cloud Connector.

Connecting applications to Remote Manager

Remote Manager provides a standard HTTP API that allows many ways to access data. HTTP APIs can be accessed by:

- A standard browser by typing in the appropriate URL
- A Java Application running on a PC or server
- A Python Application running on a PC or server
- A Python Application running on a Device
- Anything that can make standard HTTP calls

Once the data is retrieved from the server, it can be used to do calculations, display graphs, monitor something, and so on.

Data flow from devices to customer applications

Remote Manager directs data flow using a clear, efficient series of steps.

Device collects data and sends it to Remote Manager

Devices can collect data using Cloud Connector, DIA, or Smart Energy Frameworks. The formats of the data and the associated Data Streams vary depending on the framework. Cloud Connector provides the most flexibility.

- When a device is collecting data, its most efficient method of recording data involves keeping track of the changes to the data and sending changes periodically. For example, if the data is temperature, the device might only send the reading if the recorded value changes by a minimum amount, or it may only update once a day, depending on which occurs first.
- If the device is recording data from many endpoints, it sends the data in batches rather than one reading at a time.

- The device includes retry logic in case the upload fails. Failure may occur if the device upload is throttled or the server is busy, or in maintenance mode.

Data is saved to data streams in Remote Manager

Remote Manager stores Data received from devices in Data Streams. Data can be saved in more than one data stream by using replication. For example, a temperature reading could be saved in a "floor 1" data stream and replicated into a "building x" data stream.

- When Remote Manager stores numeric data, it also creates rollups that compute statistics based on time intervals. This allows users to make queries such as the hourly sum of all sodas purchased at a vending machine.
- The amount of time the data is available on the server is determined by the TTL (time to live). Raw data and rollup data use different TTLs of different lengths; rollup TTLs are longer than raw data TTLs. When the TTL of a particular set of data expires, Remote Manager automatically deletes the data. Remote Manager sets the TTL when data is stored. Default values and maximum TTLs vary depending on your account's tiered pricing model.

Customer application receives information from Remote Manager

Customer applications can receive data in several ways depending on the needs of the application:

- **Push notifications**
  The Monitor API enables customer applications to register for asynchronous (push) notification of events within Remote Manager. The registering application can choose which types of events they are interested in. For example, the application can register to receive all data stream data. The Monitor API allows the user to send notifications using HTTP or TCP connections to the client application.
- **Alarms**
  The user can configure alarms to process the data as it is saved in Remote Manager. Alarms can watch for conditions. such as data values outside of a range or missing data. When an alarm is triggered, Remote Manager can send the client notification via a monitor or email. The alarm status will also display in the web-based user interface.
- **Web service queries**
  The DataStream, DataPoint, and v1/streams web service APIs can be used to query for data. The DataStream API can return information about all data streams in your account, including metadata such as the data type and current value. The DataPoint API can return a list of timestamped values for a specific data stream, and also can include metadata such as the description and location, if included when the data was created. The v1/streams API can manage both data streams and datapoints.

Remote Manager concepts

Subscriptions

Remote Manager subscriptions control access to Remote Manager features, and the available features vary depending on the Remote Manager account edition: Platform, Standard, Premier, or Developer.

Device IDs

A device ID is a unique 16-byte number used to uniquely identify a device within Remote Manager.

Most device IDs are derived from the device MAC address, IMEI number, or ESN number. If a device does not have an assigned MAC, IMEI, or ESN, Remote Manager generates and assigns a random 16-byte number for the device ID. See System-generated device IDs for more information.

Note In resource web services, device IDs are listed as devConnectwareId elements. See the *Digi Remote Manager Programmer Guide*.

Device ID Assignments

A device ID is derived from the unique information from the device, in the order specified in the list below.

1. The Ethernet interface MAC-48. See Device IDs based on MAC addresses.
2. The 802.11 interface MAC-48. See Device IDs based on MAC addresses.
3. The cellular modem IMEI for GSM devices. See Device IDs based on GSM IMEI.
4. The cellular modem ESN (Electronic Serial Number) for CDMA devices. See Device IDs based on CDMA addresses.
5. The auto-generated format. See System-generated device IDs.

For example, if a device has an Ethernet interface and a cellular modem, the device ID is generated from the Ethernet interface. If a device contains multiple interfaces of one type (such as two Ethernet interfaces), a primary interface is selected and used as the source of the device ID.

Full-length device IDs

The full-length device ID is specified as four groups of eight hexadecimal digits separated by dashes. For example:

```
01234567-89ABCDEF-01234567-89ABCDEF
```

Abbreviated device IDs

Device IDs can also be specified in an abbreviated form, without the leading groups of zeros. The following table shows how some device IDs can be abbreviated.

Full device ID	Abbreviated forms
00000000-89ABCDEF-01234567-89ABCDEF	89ABCDEF-01234567-89ABCDEF

Full device ID	Abbreviated forms
00000000-00000000-01234567-89ABCDEF	00000000-01234567-89ABCDEF 01234567-89ABCDEF
01234567-89ABCDEF-01234567-89ABCDEF	No abbreviated form
00000000-00000000-00000000-89ABCDEF	00000000-00000000-89ABCDEF 00000000-89ABCDEF 89ABCDEF

System-generated device IDs

Remote Manager can automatically generate and assign a device ID. Generated IDs are often used for devices that do not have a unique identifier.

Here is a sample system-generated device ID:

0008cccc-eeeeeeee-vvvvvvvv-gggggggg

System-generated value	Description
cccc	Unique value set per cluster, dependent on the generated cluster ID
eeeeeeee	Typically all zeroes, but may be randomly assigned
vvvvvvvv	Represents a provision ID for the customer, currently the vendor ID
gggggggg	Randomly assigned

Device IDs based on CDMA addresses

CDMA (Code Division Multiple Access) device IDs have two addressing schemes:

- 32-bit Electronic Serial Number (ESN) scheme
- 56-bit Mobile Equipment Identifier (MEID) scheme

Both addresses can be specified in hexadecimal or decimal format.

ESN-Hex address: MM-SSSSSS
Device ID mapping: 00020000-00000000-00000000-MMSSSSSS

MEID-Hex address: RR-XXXXXX-ZZZZZZ-C
Device ID mapping: 00040000-00000000-00RRXXXX-XXZZZZZZ

Note A check digit is appended to MEID addresses. The check digit is not part of the MEID and is therefore not included in the device ID mapping.

Device IDs based on MAC addresses

Device IDs can be derived from the 48-bit MAC address.

For example:

MAC address: 112233:445566
Device ID mapping: 00000000-00000000-112233FF-FF445566

Device IDs based on GSM IMEI

Device IDs can be derived from a GSM IMEI address which consists of 14 decimal digits plus a check digit. The check digit is not officially part of IMEI. However, since modems commonly report the IMEI including check digit and it is typically listed on labels, the check digit is included in the device ID mapping.

Example IMEI: AA-BBBBBB-CCCCCC-D
Device ID mapping: 00010000-00000000-0AABBBBB-BCCCCCCD

Embedded device development

Devices manufactured by Digi International contain firmware enabled for Remote Manager. Third-party device developers can create devices that are Remote Manager enabled using development kits, such as Cloud Connector.

When a device connects to Remote Manager, the device supplies a vendor ID and device type. The vendor ID is the namespace for the vendor and the device type is a vendor-specific unique name for a device type. Because device types are vendor-specific, multiple vendors can use the same name for a device type. The device type name must be unique within the vendor only—not unique within all of Remote Manager.

For example, a device manufacturer with the vendor ID 3000 can create a device type named iVendingMachine, while another vendor with the vendor ID 3001 can create a device type of the same name. The two device types of the same name are unique because they are associated with different vendor IDs.

For more information on using third-party devices with Remote Manager, see the *Digi Cloud Connector Getting Started Guide* and *Digi Cloud Connector User Guide*.

About data services

Remote Manager data services allow you to collect and manage data from remote devices. For example, a device can send data files to the Remote Manager server, and Remote Manager temporarily caches the data files in a database. The files are stored by default for 24 days. Any files stored in the my_tasks folder are saved indefinitely.

Data collections and files

Remote Manager stores the files in collections, which are similar to folders. You can access files and collections using the Remote Manager user interface and web services.

Home collection tilde (~) character

For each user account, Remote Manager creates a home collection, which is represented by the tilde (~) character. All files for your user account are stored relative to your home directory. For example, if you created a collection named **mydata**, you can access the **mydata** collection as follows:

```
~/mydata
A collection for a device includes the Device ID.
~/00000000-00000000-00000000-12345678
```

Device information caching

To provide fast response times and reduce network bandwidth, Remote Manager caches device-related data. Some of the cached data is related to a specific device and some is meta data about groups of devices. The Remote Manager server has numerous caching mechanisms to organize and store this data. The sections below describe these mechanisms and the data they store.

Using cached data to respond to requests

By default, the cache is automatically used to satisfy requests for information. However the caller has some ability to control whether cache is used:

cache="false"

When issuing an SCI request, the caller can specify the attribute *cache="false"* on the send_message command. This attribute instructs the server to ignore the cache and always forward the request on to the device. A caller may do this if they suspect the cache is stale and it is a way to refresh the contents of the cache.

cache="only"

The caller can also specify *cache="only"* to instruct the server to provide responses from the cache only and never send them on to the device. A user can use this option if they are interested in the data if it is cached but they do not want to incur the overhead of communicating with the device.

Limiting information returned from requests

The user can control the amount of data returned from a request using the reply attribute:

reply="error"

Returns only error responses from a send_message command.

reply="none"

Does not return any responses from a send_message command.

reply="all"

Returns all responses from a send_message command.

The reply attribute only controls how much of the response data is streamed back to the user. Remote Manager inspects the reply data and updates the various data caches appropriately, regardless of the reply attribute.

Web services HTTP client applications

Remote Manager provides a Rest-style API over HTTP (or HTTPS). Users can write HTTP clients in their preferred programming language that get data from Remote Manager and use or display the data. Examples of such clients include web pages and programs written in a language such as Python or Java. These clients send requests to the Remote Manager server using standard HTTP requests. Remote Manager supports the following HTTP requests: GET, PUT, POST, and DELETE.

Authentication best practices

The Remote Manager server supports basic HTTP authentication and only valid users can access the database. To reduce the authentication overhead of multiple requests, use either an HTTP library that caches cookies or cache the cookies JSESSIONID and SID yourself.

Using a web browser

You can enter any GET request into the URL field of a web browser. Some browser plug-ins allow you to call other HTTP methods.

Using the API explorer

The Remote Manager API explorer available via the **Documentation > API Explorer** tab allows you to run any web service request, as well as export code as Python, Java, Ruby, Perl, or C## code.

Using a Python script

You can write Python scripts to send standard HTTP requests to the server. These scripts use Python libraries to handle connecting to the Remote Manager server, sending the request, and getting the reply. See the following sites for more information on Remote Manager Python open source libraries:

- https://github.com/digidotcom/python-devicecloud
- https://digidotcom.github.io/python-devicecloud/

Using a Java program

You can send HTTP requests to the server through a Java program. Below is a code snippet of an HTTP POST of an SCI request written in Java.

```
import java.io.IOException;
import java.io.InputStream;
```

```
import java.io.OutputStreamWriter;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Scanner;
/*
 * Can replace this with any base 64 encoder for basic authentication. For java 6
installations on Sun's JRE you can use "sun.misc.BASE64Encoder"
 * however this will not work in some installations (using OpenJDK). Java mail
(javax.mail.internet.MimeUtility) also contains a Base 64 encoder in Java 6.
 * A public domain version exists at
http://iharder.sourceforge.net/current/java/base64/
 */
import org.apache.commons.codec.binary.Base64;
/**
* This is a stub class with a main method to run a Remote Manager web service.
*/
public class WebServiceRequest {
 /**
 * Run the web service request
 */
 public static void main(String[] args) {
    try {
          // Create url to Remote Manager server for a given web service request
          URL url = new URL("https://remotemanager.digi.com/ws/sci");
          HttpURLConnection conn = (HttpURLConnection)url.openConnection();
          conn.setDoOutput(true);
          conn.setRequestMethod("POST");
          // replace with your username/password
          String userpassword = "YourUsername:YourPassword";
          // can change this to use a different base64 encoder
          String encodedAuthorization = Base64.encodeBase64String
(userpassword.getBytes()).trim();
          conn.setRequestProperty("Authorization", "Basic "+
encodedAuthorization);
          // Send data to server
          conn.setRequestProperty("Content-Type", "text/xml");
          OutputStreamWriter out = new OutputStreamWriter(conn.getOutputStream());
          out.write("<sci_request version=\"1.0\"> \r\n");
          out.write(" <send_message> \r\n");
          out.write(" <targets> \r\n");
          out.write(" <device id=\"00000000-00000000-00000000-00000000\"/>\r\n");
          out.write(" </targets> \r\n");
          out.write(" <rci_request version=\"1.1\">\r\n");
          out.write(" <query_state/>\r\n");
          out.write(" </rci_request>\r\n");
          out.write(" </send_message>\r\n");
          out.write("</sci_request>\r\n");
          out.close();
          // Get input stream from response and convert to String
          conn.disconnect();
          conn.setDoInput(true);
          InputStream is = conn.getInputStream();
          Scanner isScanner = new Scanner(is);
          StringBuffer buf = new StringBuffer();
          while(isScanner.hasNextLine()) {
             buf.append(isScanner.nextLine() +"\n");
          }
          // Output response to standard out
          String responseContent = buf.toString();
```

```
            System.out.println(responseContent);
        } catch (IOException e) {
            // Print any exceptions that occur
            e.printStackTrace();
        }
    }
}
```

Web services conventions and versioning

Remote Manager has two versions of supported web services:

- **Pre-version 1 APIs:** Original set of Remote Manager APIs released before API versioning was introduced.
- **Version 1 APIs:** Set of APIs each of which includes **v1** as the first element in the resource URI.

Pre-version 1 and version 1 APIs have different defaults for media formats: Pre-version 1 APIs default to XML for the media format; Version 1 APIs default to JSON for the media format.

URL specifications

Remote Manager web services APIs are RESTful in nature. Each URL relates to a specific Remote Manager resource or list of resources.

For example, the following URL retrieves a list of devices:

```
https://remotemanager.digi.com/ws/v1/devices/inventory
```

The following URL retrieves a single device with device ID 00000000-00000000-00409DFF-FF038457:

```
https://remotemanager.digi.com/ws/v1/devices/inventory/00000000-00000000-
00409DFF-FF038457
```

And the following URL requests a list of alarm templates:

```
https://remotemanager.digi.com/ws/AlarmTemplate
```

Default media format

The default media format for Remote Manager APIs depends on the API version:

- Pre-version 1 APIs default to XML as the media format.
- Version 1 APIs default to JSON as the media format.

Note Some browsers set the Accept header to XML. If you unexpectedly receive an XML response from a version 1 API request, your browser Accept header may be set to XML. To force JSON output regardless of the browser Accept header, include the JSON (.json) extension on your request. For example:

```
https://remotemanager.digi.com/ws/v1/devices/inventory.json
```

JSON formatting

In JSON, resources are represented as JSON objects and lists are represented as arrays.

XML formatting

In XML, resources are wrapped with a resource element (for example, <device>) and lists are wrapped in a <list> element. In addition, all results returned in response data are wrapped in a <results> element as the root.

Supported HTTP headers

Remote Manager APIs support standard HTTP headers. The following request and response headers are handled specially or are custom headers.

Request headers

The following request headers are supported:

- **Accept:** Indicates the expected content type for a request, as well as the content type for the response for all HTTP methods. At present, application/JSON and application/XML are supported. If you do not specify an accept header or you specify a header with an unsupported type, the application/JSON header is used.

Response headers

The following response headers are supported:

- **Delete-Count:** Custom header set with the number of resources deleted on a DELETE request. If the number of deleted cannot be determined, then this header is not set. For example, deleting all history (data points) for a stream does not return the deleted count.
- **Location:** Set to the URL of the created/updated resource of a PUT or POST. For multiple resources, the location header is set to the last resource successfully created/updated.

Pagination options

Requests that return multiple resources are paginated. The default and maximum page size is 1000. You can request smaller pages using the size parameter.

Pagination parameters

Paged responses include:

- **next_uri:** URI value that can be used to request the next page of data. No value is returned if there are no more pages available.
- **count:** Number of resources returned.
- **size:** Size requested.
- **cursor:** Placeholder that can be used to request the next page of data. The cursor is included in the next_uri value, which is not returned if there are no more pages available.

Depending on the request type, the response may also contain the following values:

- **start_time:** Start time for time-series data requests.
- **end_time:** End time for time-series data requests.

Using pagination parameters

To use page parameters to paginate responses:

1. Request the first page of data.
2. Use the **next_uri** value to request the next page.
3. Continue to use the **next_uri** value returned in a page to get the next page.
4. When no **next_uri** is returned, you have retrieved all the data.

CRUD conventions

The following CRUD (Create/Read/Update/Delete) conventions are used:

Action	Create	Read	Update	Delete
HTTP Verb	POST/PUT*	GET	PUT	DELETE

*Use POST to create a resource that has a system-generated ID; use PUT to create a resource with a known ID (that is, an ID composed of known composite values).

Post operation

HTTP POST is used to add resources to your account.

Format

/ResourcePath

Request content

XML or JSON representation of a resource OR a list of resources in the format `<list>. . .</list>`

Example request content:

```
<DeviceCore>
    <devMac>00:40:9D:22:22:21</devMac>
</DeviceCore>
```

Example request content with a list:

```
<list>
   <DeviceCore>
      <devMac>00:40:9D:22:22:22</devMac>
   </DeviceCore>
   <DeviceCore>
      <devMac>00:40:9D:22:22:23</devMac>
   </DeviceCore>
</list>
```

Return codes

201 (Created) A new resource (or list of resources) was created.

207 (Multi-status) A list was passed in but not all were created.

400 (Bad request) The request is invalid.

401 (Unauthorized) The user ID/password is invalid.

403 (Forbidden) Access to the resource is not authorized.

429 (Too many requests) The request has been throttled. Wait and try again.

500 (Internal server error) The request cannot be handled due to a server error. Wait and try again.

Response header

Location contains the URI for a created resource (last resource created for a list).

Return content

XML or JSON document with a root result element containing a location element for each resource created and any errors encountered.

Get operation

HTTP GET is used to retrieve a specific resource by ID or a list of resources.

Format

/ResourcePath gets a list of all resources in the account matching the authorization credentials

/ ResourcePath /.json gets a list of all resources in JSON format

/ ResourcePath /.xls gets a list of all resources in Excel format

/ ResourcePath /ID gets a resource for the specified ID

/ ResourcePath /ID.json gets a resource for the specified ID in JSON format

/ ResourcePath /ID.xls gets a resource for the specified ID in Excel format

Query parameters

start - the record number to start the results from

size - the number of records to return

condition - a query where condition is used to filter the results

orderby - a column used to sort the results

Request headers

Name: Accept Value: application/json Effect: Returns a JSON view of the resource

Name: Accept Value: application/xml Effect: Returns an XML view of the resource (default)

Name: Accept Value: application/vnd.ms-excel Effect: Returns an excel view of the resource

Name: Authorization Value: Basic {Base64 encoded password} Effect: Authorizes resource access

Note We recommend you use the **Accept-Encoding: gzip, deflate** request header to instruct the server to return the data compressed with a return header of **Transfer-Encoding: gzip**. This improves GET performance and is generally transparent to most client libraries.

Return codes

200 (OK)

400 (Bad request) The request is invalid.

401 (Unauthorized) The user ID/password is invalid.

403 (Forbidden) Access to the resource is not authorized.

429 (Too many requests) The request has been throttled. Wait and try again.

500 (Internal server error) The request cannot be handled due to a server error. Wait and try again.

Return content

The return content is an XML document with a root result element containing one or more elements of the resource type and any errors encountered; or a JSON document with results and errors. Any elements that have no content (essentially null) are not returned.

The returned content includes a header to help the user make multiple calls.

```
<result>
  <!-- total number of resources that match the condition -->
<resultTotalRows>13</resultTotalRows>
  <!-- the record number of the first result -->
<requestedStartRow>11</requestedStartRow>
  <!-- the number of results returned -->
<resultSize>2</resultSize>
  <!-- the number of results requested -->
<requestedSize>2</requestedSize>
  <!-- the remaining number of resources -->
<remainingSize>0</remainingSize>
  <!-- ... List of the resources -->
```

Examples

http://<hostname>/ws/DeviceCore
Returns all devices in the account matching the authorization credentials

http://<hostname>/ws/DeviceCore/32
Returns the device information matching the device where ID=32 (ID is an auto-generated number in Remote Manager)

http://<hostname>/ws/DeviceCore?start=201&size=200
Returns 200 records starting with record 201

http://<hostname>/ws/DeviceCore?condition=devRecordStartDate>'2010-01-17T00:00:00Z'
Returns all devices added after midnight Jan 17th, 2010

http://<hostname>/ws/DeviceCore/?condition=devConnectwareId='00000000-00000000-00409DFF-FF123456'
Returns the record for device ID "00000000-00000000-00409DFF-FF123456"

Sample result of http://<hostname>/ws/DeviceCore?start=11&size=2 request:

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
   <resultTotalRows>13</resultTotalRows>
   <requestedStartRow>11</requestedStartRow>
   <resultSize>2</resultSize>
   <requestedSize>2</requestedSize>
   <remainingSize>0</remainingSize>
   <DeviceCore>
      <id>
         <devId>155</devId>
         <devVersion>0</devVersion>
      </id>
      <devRecordStartDate>2010-06-25T21:28:00Z</devRecordStartDate>
      <devMac>00:40:9D:3D:71:A6</devMac>
      <devConnectwareId>00000000-00000000-00409DFF-FF3D71A6</devConnectwareId>
      <cstId>3</cstId>
      <grpId>3</grpId>
      <devEffectiveStartDate>2010-06-25T21:28:00Z</devEffectiveStartDate>
      <devTerminated>false</devTerminated>
      <dvVendorId>0</dvVendorId>
      <dpDeviceType>ConnectPort X2</dpDeviceType>
      <dpFirmwareLevel>34209795</dpFirmwareLevel>
      <dpFirmwareLevelDesc>2.10.0.3</dpFirmwareLevelDesc>
      <dpRestrictedStatus>0</dpRestrictedStatus>
      <dpLastKnownIp>10.20.1.161</dpLastKnownIp>
      <dpGlobalIp>10.20.1.161</dpGlobalIp>
      <dpConnectionStatus>1</dpConnectionStatus>
      <dpLastConnectTime>2010-06-28T13:35:00Z</dpLastConnectTime>
      <dpContact />
      <dpDescription>EMS - Aux gateway #2 - Test certificate</dpDescription>
      <dpLocation>Jeff's office</dpLocation>
      <dpPanId>0x134f</dpPanId>
      <xpExtAddr>00:13:a2:00:40:5c:0a:ba</xpExtAddr>
      <dpServerId>ClientID[3]</dpServerId>
   </DeviceCore>
   <DeviceCore>
      <id>
         <devId>156</devId>
         <devVersion>0</devVersion>
      </id>
      <devRecordStartDate>2010-06-25T20:46:00Z</devRecordStartDate>
      <devMac>00:40:9D:29:5B:4C</devMac>
      <devConnectwareId>00000000-00000000-00409DFF-FF295B4C</devConnectwareId>
      <cstId>3</cstId>
      <grpId>3</grpId>
      <devEffectiveStartDate>2010-06-25T20:46:00Z</devEffectiveStartDate>
      <devTerminated>false</devTerminated>
      <dvVendorId>0</dvVendorId>
      <dpDeviceType>Digi Connect WAN VPN</dpDeviceType>
      <dpFirmwareLevel>34014219</dpFirmwareLevel>
      <dpFirmwareLevelDesc>2.7.2.11</dpFirmwareLevelDesc>
      <dpRestrictedStatus>0</dpRestrictedStatus>
      <dpLastKnownIp>10.20.1.144</dpLastKnownIp>
      <dpGlobalIp>10.20.1.144</dpGlobalIp>
      <dpConnectionStatus>1</dpConnectionStatus>
      <dpLastConnectTime>2010-06-28T13:35:00Z</dpLastConnectTime>
      <dpDescription>Test device</dpDescription>
      <dpLocation />
      <dpServerId>ClientID[3]</dpServerId>
```

```
   </DeviceCore>
</result>
```

Put and Post operations

HTTP PUT is used to update a resource at the specified location. If a resource has an ID containing composite values rather than generated, it can be created using a PUT. A resource that has an ID that is generated by the database must be created using POST.

Format

/ResourcePath/ID

Example

```
/NetworkInterface/1
```

Request content

XML or JSON representation of an updated resource.

An ID must be specified either in the path or in the content. If an ID is in both places, they must match.

Return codes

200 (OK)

201 (Created) A new resource (or list of resources) was created. (POST)

207 (Multi-status) A list was passed in and not all were created. Specific errors are included in the response body. (POST)

400 (Bad request) The request is invalid.

401 (Unauthorized) The user ID/password is invalid.

403 (Forbidden) Access to the resource is not authorized.

429 (Too many requests) The request has been throttled. Wait and try again.

500 (Internal server error) The request cannot be handled due to a server error. Wait and try again.

Return content

XML or JSON document with a root result element containing any errors encountered.

Delete operation

HTTP DELETE is used to delete a resource from your account.

Format

/ResourcePath/ID

Example

```
http://<hostname>/DeviceCore/1
```

Return codes

200 (OK)

204 (No Content) The delete was successful and no content is being returned.

400 (Bad request) The request is invalid.

401 (Unauthorized) The user ID/password is invalid.

403 (Forbidden) Access to the resource is not authorized. You may need a subscription.

429 (Too many requests) The request has been throttled. Wait and try again.

500 (Internal server error) The request cannot be handled due to a server error. Wait and try again.

Return content

XML or JSON document with a root result element containing any errors encountered.

URL encoding (percent encoding)

URLs cannot contain spaces or non-ASCII characters. Use URL encoding to convert non-ASCII characters into a format that can be transmitted over the internet. URL encoding replaces non-ASCII characters with a percent sign (%) followed by two hexadecimal digits.

For example, when including a timezone such as +01:00 on a DataPoint request, URL encode the request as follows:

```
https://remotemanager.digi.com/ws/DataPoint/00010000-00000000-03515790-
56477597/ain1/val?startTime=2016-11-01T16:00:03.0000000%2B01:00
```

For more information on URL encoding, see HTML URL Encoding Reference.

Best practice: Use compound queries

Most web services sample requests show simple requests to best demonstrate how each web service API works. However, you may need to set up compound queries. Here are two examples that demonstrate how compound queries operate, along with the corresponding expected results:

The following example:

```
/ws/v1/streams/history/dia/channel/00000000-00000000-00409DFF-
FF72E822/system0/cpu_utilization?start_time=2017-01-10T17:00:00.000&end_
time=2017-01-10T18:00:00.000&order=desc
```

Returns data points that occurred within a one-hour period in descending order.

The following example:

```
/ws/DeviceCore?condition=dpDeviceType='ConnectPort X2e SE' AND
dpLastConnectTime>'2017-01-27T00:00:00.000Z'
```

Returns all the X2e devices that connected after January 27th at midnight GMT.

Note Modern browsers MIME-encode spaces (%20) and punctuation, such as single quote (%27) on your behalf, but many programming languages do not. In those cases you must put them in proper MIME format in your code.

Best practice: Wrap multiple queries into a single request

To perform multiple tasks, wrap all the tasks within a single request. For example, to perform the following three tasks:

1. Get a list of files in Python
2. Look up the device info
3. Look up system settings

Wrap the three tasks into a single request as follows:

```
<sci_request version="1.0">
   <send_message cache="false">
      <!-- list targets for query -->
      <targets>
         <device id="00000000-00000000-00000000-00000000" />
      </targets>
      <rci_request version="1.1">
         <!-- Request python files -->
         <do_command target="file_system">
            <ls dir="/WEB/python" />
         </do_command>
         <!-- Lookup device state -->
         <query_state>
            <device_info />
         </query_state>
         <!-- Return system settings -->
         <query_setting>
            <system />
         </query_setting>
      </rci_request>
   </send_message>
</sci_request>
```

Best practice: Reuse HTTP sessions

When a web service request is made, the request must send credentials via HTTP basic authentication. Subsequent calls can use the same session created on the first request. Using one HTTP session rather than multiple sessions (one for each request) improves performance because you need not repeat authentication.

Using the same session, however, does add complexity to the client code because the code must handle HTTP cookie management.

Web services reference

Web service	GET	POST	PUT	DELETE	Description
Alarm	✔	✔	✔	✔	Alarm and alarm properties
AlarmStatus	✔		✔		Alarm status information
AlarmStatusHistory	✔				Alarm status history information
AlarmTemplate	✔				Alarm templates
CarrierAuth	✔	✔	✔	✔	Carrier authorization
DataPoint	✔	✔	✔	✔	Data point management
DataStream	✔	✔	✔	✔	Data stream management
DeviceCore	✔	✔	✔	✔	Device and selected properties
DeviceInterface	✔				Device and network interface properties
DeviceMetaData	✔	✔	✔	✔	Device descriptor data
DeviceVendor	✔	✔			Embedded device developer information
DeviceVendorSummary	✔				Device vendor properties
FileData	✔	✔	✔	✔	Data files

Web service	GET	POST	PUT	DELETE	Description
FileDataCore	✔				Data files information
FileDataHistory	✔				Data file history information
Group	✔				Device groups
Monitor	✔	✔	✔	✔	Monitors management
NetworkInterface	✔	✔	✔	✔	Device modem list
NetworkInterfaceSubscriptionCore	✔				Carrier subscription information
openapi	✔				Experimental feature that provides OpenAPI summary for the v1/devices and v1/streams APIs
Schedule	✔	✔	✔	✔	Operations, tasks, and schedules
SCI (Server command interface)	✔	✔	✔	✔	Device-specific commands
security		✔			Device connection password
Task	✔				Task management
v1/alerts	✔	✔	✔	✔	Alert properties
v1/devices	✔	✔	✔	✔	Device and selected properties
v1/events	✔				Event properties.
v1/groups	✔	✔	✔	✔	Group properties
v1/health_configs	✔		✔	✔	Device health configurations
v1/jobs	✔	✔	✔	✔	Job properties

Web service	GET	POST	PUT	DELETE	Description
v1/metadata	✔				Device descriptors
v1/monitors/history	✔				Query monitors
v1/reports	✔				Device reports
v1/settings	✔				Device settings
v1/streams	✔	✔	✔	✔	Data stream and points management
v1/users	✔	✔	✔	✔	User properties
XbeeAttributeCore	✔				XBee attributes
XbeeAttributeFull	✔				XBee attributes
XbeeClusterCore	✔				XBee cluster information
XbeeCore	✔		✔		XBee nodes and properties

openapi

Use the **openapi** web service to get summary information on the **ws/v1/devices** and **ws/v1/streams** APIs.

Note This service is experimental and may be changed or removed in a future release.

URI

```
https://<hostname>/ws/openapi
```

Formats

HTTP method	Format	Description
GET	/ws/openapi	Get a summary of the ws/v1/devices and ws/v1/streams APIs.

Query language for v1 APIs

Some Remote Manager v1 APIs include a **query** parameter for GET requests. Using the query parameter, you can build complex expressions for selecting Remote Manager objects.

Query language summary

- Similar in concept to SQL or other query languages. Use conditions and operators based on field types.
- Single quoted text literals 'TheText' .
- Text escape for quote character is the quote: 'isn''t difficult' .
- Numeric literals support 0x prefix for hex.
- Relative values from "now' for timestamp values. For example, -10s for 10 seconds ago.
- Text-based comparisons are case insensitive.
- Use the **and** and **or** keywords as well as parenthesis to group simple conditions into more complex expressions.
- Use the special value keyword **empty** to represent empty string, null, and unset.

Note Although the query parameter in each API provides the same query expression capability, the fields that you can query depend on the fields returned for objects of that API. For example, a query using the **ws/v1/devices/inventory** API specifies device fields, while a query using the **ws/v1/alerts/inventory** APIs specifies alert fields.

Note Be sure to correctly URL encode the query expression (for example space encodes to **%20** in a URL parameter value.

Specify values in query conditions

The syntax for specifying literal values varies depending on the type of literal value. Not all syntaxes from other query languages are supported.

Value Type	Description
String Enumerated String Group	Specify these values using single quotes. If the value contains a single quote, specify two single quotes. Enumerated strings cannot accept all values, for example `connection_status` can only be `connected` or `disconnected`. For example: ■ connection_status = 'connected' ■ name = 'Fred''s Device' ■ group = 'TheGroup'

Value Type	Description
Numeric	Specify these values using a normal decimal number notation or a hex number notation. ■ value = 0 ■ value = -3.14 ■ value = 0x10
Timestamp	Specify these values using a relative time notation where only negative values are supported. Use a one letter suffice to indicate time units: s for seconds, m for minutes, h for hours, d for days (exactly 24 hours) and w for weeks (exactly 7 days). ■ last_updated > -60m ■ last_connect >= -24h
Geoposition	Specify a bounding box value. The geoposition can be within or outside of the bounding box. The bounding box consists of four coordinates of the form: [Southwest longitude, Southwest latitude, Northeast longitude, Northwest latitude]. For example, to select items approximately within the continental United States: ■ geoposition within [125.0, 25.0, 65.0, 50.0]

Specify operators in query conditions

The following table summarizes Remote Manager v1 query language conditions.

Operator	Permitted Types	
= <>	Enumerated String Enumerated Value Group Tags Timestamp	Exact equality or inequality. For a tag, indicates the presense or absense of the tag on the item. For a group, indicates the full group path.
< <= >= >	String Numeric Timestamp	The string, number or timestamp sorts before or after the value.
contains	String Tag	The String contains the specified substring. Any tag contains the specified substring.

Operator	Permitted Types	
startswith endswith	String Tag Group	The string starts with or ends with the specified value. Any tag starts with or ends with the specified value. For a group, targets the full group path. For example, group startswith 'test/' targets any device in the root/test group and all subgroups, while group startswith 'test' targets any device in the root/test* groups and any subgroups.
within outside	Geoposition	The geoposition is within or outside of a bounding box. For example, to select items approximately within the continental United Statess, query for geoposition within [125.0, 25.0, 65.0, 50.0]

Example Queries

- Complex Queries
  - `query=group startsWith '/NorthWest' and (connection_status = 'disconnected' or signal_percent < 20)` - find any devices in the /Northwest group and any subgroups that are either disconnected or have a low signal strength
  - `query=tags = 'important' and (health_status = 'error' or health_status = 'warning')` - Find any devices that have the 'important' tag and are in an error or warning health status
  - `query=last_connect = empty` - Find any devices whose last connect value is unset (have never connected).
- Group Queries
  - `query=group = '/test'` - query full group path, so matches any device in group '/test' and ignores any subgroups.
  - `query=group startsWith 'test/'` - query full group path, so matches any device in the test root group and any subgroups.
  - `query=group startsWith 'test'` - query full group path, so matches any device in any root group whose name starts with 'test' and all subgroups.
  - `query=group endsWith '/leaf'` - query full group path, so matches any device in any path that ends with the group name 'leaf'.
- Tag Queries
  - `query=tags = empty` - matches any device having no tags.
  - `query=tags <> empty` - matches any device having any tags.
  - `query=tags = 'sensor'` - matches any device having a tag 'sensor'.

- `query=tags <> 'sensor'` - matches any device having no tag 'sensor'.
- `query=tags contains 'ns'` - matches any device having any tag containing 'ns'.
- `query=tags startsWith 'sens'` - matches any device having any tag that starts with 'sens'.
- `query=tags endsWith 'or'` - matches any device having any tag that ends with 'or'.

■ Geoposition Queries

- `query=geoposition within [ 125.0, 25.0, 65.0, 50.0 ]` - matches any device with coordinates within the specified bounding box (approximately the continental United States).
- `query=geoposition outside [ 125.0, 25.0, 65.0, 50.0 ]` - matches any device with coordinates outside the specified bounding box (approximately the continental United States).

Deprecated APIs

The following APIs have been deprecated and should not be used in new code. For compatibility, deprecated APIs will be supported for a limited time, but code containing the deprecated APIs should be modified to use a supported API as soon as possible.

Resource path	Migrate to use
CarrierSubscription	`/ws/v1/streams/inventory?category=carrier`
CarrierUsage	`/ws/v1/streams/history/{device_id}/carrier/{sim_id}/usage/{usage_id}`
data	`ws/FileData`
DiaChannelDataFull	`ws/DataPoint/dia/channel/<Device Id>/<instance>/<channel>`
DiaChannelDataHistoryFull	`ws/DataPoint/dia/channel/<Device Id>/<instance>/<channel>`
XbeeAttributeDataCore	`ws/DataPoint/se/attr/<Device Id>/<XBee Address>/<Endpoint Id>/<Cluster Type>/<Cluster Id>/<Attribute Id>`
XbeeAttributeDataFull	`ws/DataPoint/se/attr/<Device Id>/<XBee Address>/<Endpoint Id>/<Cluster Type>/<Cluster Id>/<Attribute Id>`
XbeeAttributeDataHistoryCore	`ws/DataPoint/se/attr/<Device Id>/<XBee Address>/<Endpoint Id>/<Cluster Type>/<Cluster Id>/<Attribute Id>`
XbeeAttributeDataHistoryFull	`ws/DataPoint/se/attr/<Device Id>/<XBee Address>/<Endpoint Id>/<Cluster Type>/<Cluster Id>/<Attribute Id>`

XbeeAttributeReportingCore	Use the following SCI commands:
	`start_reports`
	`get_local_reporting_configurations`

XbeeEventDataCore	`ws/DataPoint/se/event/<Device Id>/<XBee Address>/<Endpoint Id>/<Cluster Type>/<Cluster Id>/<Attribute Id>`
XbeeEventDataFull	`ws/DataPoint/se/event/<Device Id>/<XBee Address>/<Endpoint Id>/<Cluster Type>/<Cluster Id>/<Attribute Id>`

Resource path	Migrate to use
XbeeEventDataHistoryCore	`ws/DataPoint/se/event/<Device Id>/<instance>/<channel>`
XbeeEventDataHistoryFull	`ws/DataPoint/se/event/<Device Id>/<XBee Address>/<Endpoint Id>/<Cluster Type>/<Cluster Id>/<Attribute Id>`

Alarm

Use the Alarm web service to create, list, or update alarms within your Remote Manager account. The maximum number of alarms per account is 16. When creating an alarm, you must specify an alarm template on which to base the alarm. See also the AlarmTemplate web service and Alarm template types.

URI

```
http://<hostname>/ws/Alarm
```

Formats

HTTP method	Format	Description
GET	/ws/Alarm	Get a list of all alarms.
GET	/ws/Alarm{almId}	Get details for a specific alarm.
POST	/ws/Alarm/{almtId}	Create a new alarm based on an existing alarm template.
PUT	/ws/Alarm/{almId}	Update an existing alarm.
DELETE	/ws/Alarm/{almId}	Delete an alarm.

Elements

almId

Remote Manager identifier for the alarm.

cstId

Remote Manager identifier for the customer.

almtID

System-generated identifier for an alarm template. To get a list of available alarm template, use the AlarmTemplate web service.

grpId

Remote Manager identifier for the customer group.

almName

Name of the alarm.

almDescription

Description of the alarm.

almEnabled

Boolean value that indicates whether the alarm is enabled.

Value	Description
true	Alarm is enabled.
false	Alarm is disabled.

almPriority

Keyword that indicates the user-defined alarm priority: high, medium, or low.

almScopeConfig

Specifies the resource scope for the alarm. Scope options include:

Scope	Description
Group	Applies the alarm to the specified group indicated by the full group path.
Device	Applies the alarm to the specified device ID. For example, 00000000-00000000-00000000-00000000.
XbeeNode	Applies the alarm to the specified XbeeNode extended address. For example, 00:13:A2:00:00:00:00:00.
Resource	Applies the alarm to a data stream path or pattern. You can use the wildcard charater asterisk (*) to match any element in the data stream path. For example, dia/channel/*/lth/temp matches all the lth temperature reading for all devices.
Global	Applies the alarm at the customer level to monitor all instances of an entity. For example, you can create an alarm to monitor the total number of web services calls for your account. This option is available for subscription usage alarms only.

See almScopeConfig for the required XML structure for almScopeConfig.

almRuleConfig

Specifies the rule configurations for an alarm:

Rule configuration	Description
FireRule	A list of variables that specify the condition for firing the alarm.
ResetRule	A list of variables that specify the condition for resetting the alarm.

By default, all alarms reset automatically. You can disable automatic reset by passing the enabled = false attribute for the ResetRule element. For example, <ResetRule enabled = "false">.

See almRuleConfig for the required XML structure for almRuleConfig.

See Alarm template types for a list of available fire and reset variables for each alarm template type.

almRuleConfig

Use the following XML structure for almRuleConfig.

```
<almRuleConfig>
    <Rules>
        <FireRule name="fireRule1">
            <Variable name="operator" value="&gt;"/>
            <Variable name="thresholdvalue" value="1"/>
            <Variable name="timeUnit" value="seconds"/>
            <Variable name="timeout" value="10"/>
            <Variable name="type" value="double"/>
        </FireRule>
        <ResetRule name="resetRule1">
            <Variable name="type" value="double"/>
            <Variable name="thresholdvalue" value="1"/>
            <Variable name="operator" value="&lt;="/>
            <Variable name="timeUnit" value="seconds"/>
            <Variable name="timeout" value="10"/>
        </ResetRule>
    </Rules>
</almRuleConfig>
```

Note By default, all alarms reset automatically. You can disable automatic reset by passing the enabled = false attribute for ResetRule element; for example, <ResetRule enabled = "false">.

almScopeConfig

Use the following XML structure for almScopeConfig.

```
<almScopeConfig>
    <ScopingOptions>
        <Scope name="Resource" value="Weather/USA/*/Minneapolis"/>
    </ScopingOptions>
</almScopeConfig>
```

Note You can specify only one <ScopingOption> per <almScopeConfig> element; and you can specify only one <Scope> per <ScopingOptions>. To specified multiple <scopes> for an alarm, use multiple <almScopeConfig> statements.

Example: List all alarms

The following example shows how to list all alarms for your account.

Request

```
GET ws/Alarm
```

Response

The sample result shows the result header and three alarms.

```
<result>
    <resultTotalRows>3</resultTotalRows>
    <requestedStartRow>0</requestedStart/row>
    <resultSize>3</resultSize>
    <requestedSize>1000</requestedSize>
    <remainingSize>0<remainingSize>
```

```
   <Alarm>
       <almId>142</almId>   <!-- alarm #1 -->
       <cstId>2</cstId>
       <almtId>4</almtId>
       <grpId>2</grpId>
       <almName>Device Excessive Connections</almName>
       <almDescription>Detects devices with an excessive number of
         connection attempts</almEnabled>
       <almPriority>0</almPriority>
       <almEnabled>true</almName>
       <almScopeConfig>
          <ScopingOptions>
               <Scope name="Group" value="/CUS0000001_Digi_International/" />
          </ScopingOptions>
       </almScopeConfig>
       <almRuleConfig>
          <Rules>
               <FireRule name="fireRule1">
                  <Variable name="disconnectWindow" value="5" />
                  <Variable name="disconnectCount" value="5" />
               </FireRule>
               <ResetRule name="resetRule1>
                  <Variable name="reconnectWindow" value="5" />
               </ResetRule>
          </Rules>
       </almRuleConfig>
   </Alarm>
```

```
   <Alarm>
       <almId>151</almId>   <!-- alarm #2 -->
       <cstId>2</cstId>
       <almtId>2</almtId>
       <grpId>2</grpId>
       <almName>Device Offline</almName>
       <almDescription>Detects when a device disconnects from Remote Manager and fails
          to reconnect within the specified time.</almEnabled>
       <almPriority>1</almPriority>
```

```
        <almEnabled>true</almName>
        <almScopeConfig>
           <ScopingOptions>
                <Scope name="Group" value="/CUS0000001_Digi_International/" />
           </ScopingOptions>
        </almScopeConfig>
        <almRuleConfig>
           <Rules>
                <FireRule name="fireRule1">
                   <Variable name="reconnectWindowDuration" value="10" />
                </FireRule>
                <ResetRule name="resetRule1 />
           </Rules>
        </almRuleConfig>
    </Alarm>
```

```
    <Alarm>
        <almId>152</almId>   <!-- alarm #3 -->
        <cstId>2</cstId>
        <almtId>1</almtId>
        <grpId>2</grpId>
        <almName>System Alarm</almName>
        <almDescription>Detects when system alarm condistions occur.</almEnabled>
        <almPriority>0</almPriority>
        <almEnabled>true</almName>
        <almScopeConfig>
        </almScopeConfig>
        <almRuleConfig>
        </almRuleConfig>
    </Alarm>
</result>
```

Example: Get details for an alarm

The following example shows how to get details for a sample alarm with an alarm ID of 3035.

Request

```
GET ws/Alarm/3035
```

Response

```
<result>
  <resultTotalRows>1</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>1</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <Alarm>
    <almId>3035</almId>
    <cstId>2</cstId>
    <almtId>2</almtId>
    <grpId>11959</grpId>
    <almName>Device Offline</almName>
    <almDescription>Detects when a device disconnects from Remote Manager and
fails to reconnected within the specified time</almDescription>
    <almEnabled>true</almEnabled>
    <almPriority>2</almPriority>
    <almScopeConfig>
      <ScopingOptions>
        <Scope name="Group" value="/CUS000000_Digi_Test/PW_Test/"/>
      </ScopingOptions>
    </almScopeConfig>
    <almRuleConfig>
      <Rules>
        <FireRule name="fireRule1">
          <Variable name="reconnectWindowDuration" value="1"/>
        </FireRule>
        <ResetRule name="resetRule1"/>
      </Rules>
    </almRuleConfig>
  </Alarm>
</result>
```

Example: Create data point condition alarm

The following sample shows how to create a data point condition alarm that fires when the outside temperature data point is less than than 10 degrees Fahrenheit below zero.

Note In this example, the alarm template ID for the data point condition alarm is 9 (almtId=9). To find the almtId for an alarm type, send a GET ws/AlarmTemplate request to get a list of all available alarm templates.

```
PUT ws/Alarm
```

```
<Alarm>
   <almtId>9</almtId>   <!-- Datapoint Condition Alarm -->
   <almName>Minneapolis Temperature</almName>
   <almDescription>Fire when it gets extremely cold.</almDescription>
   <almScopeConfig>
      <ScopingOptions>
         <Scope name="Resource" value="temperature/MN/Minneapolis" />
      </ScopingOptions>
   </almScopeConfig>
   <almRuleConfig>
      <Rules>
         <FireRule>
            <Variable name="thresholdValue" value="-10" />
            <Variable name="timeUnit" value="minutes" />
            <Variable name="type" value="numeric" />
            <Variable name="timeout" value="10" />
            <Variable name="operator" value="&lt;" />
         </FireRule>
         <ResetRule>
            <Variable name="thresholdValue" value="-10" />
            <Variable name="timeUnit" value="minutes" />
            <Variable name="type" value="numeric" />
            <Variable name="timeout" value="10" />
            <Variable name="operator" value="&gt;" />
         </ResetRule>
      </Rules>
   </almRuleConfig>
</Alarm>
```

Example: Create a DIA channel data point condition alarm

The following sample shows how to create a DIA channel data point condition alarm that fires when the helium level in an MRI gets low.

Note In this example, the alarm template ID for the DIA channel data point condition alarm is 6 (almtId=6). To find the almtId for an alarm type, send a GET ws/AlarmTemplate request to get a list of all available alarm templates.

```
PUT ws/Alarm
```

```
 <Alarm>
      <almtId>6</almtId>   <!-- Dia Channel DataPoint Condition Alarm -->
      <almName>Low Helium</almName>
      <almDescription>Fires when the helium level in the MRI gets
low</almDescription>
      <almScopeConfig>
         <ScopingOptions>
            <Scope name="Group" value="CUS001_ABC/Test/" />
         </ScopingOptions>
      </almScopeConfig>
      <almRuleConfig>
         <Rules>
            <FireRule>
               <Variable name="thresholdValue" value="10" />
               <Variable name="channelName" value="helium" />
               <Variable name="instanceName" value="mri" />
               <Variable name="timeUnit" value="seconds" />
               <Variable name="type" value="numeric" />
               <Variable name="timeout" value="5" />
               <Variable name="operator" value="&lt;" />
            </FireRule>
            <ResetRule>
               <Variable name="thresholdValue" value="10" />
               <Variable name="channelName" value="helium" />
               <Variable name="instanceName" value="mri" />
               <Variable name="timeUnit" value="seconds" />
               <Variable name="type" value="numeric" />
               <Variable name="timeout" value="5" />
               <Variable name="operator" value="&gt;" />
            </ResetRule>
         </Rules>
      </almRuleConfig>
   </Alarm>
```

Example: Create a smart energy missing data point alarm

The following sample shows how to create a smart energy missing data point alarm that fires when the devices do not report smart energy data.

Note In this example, the alarm template ID for the smart energy missing data point alarm is 12 (almtId=12). To find the almtId for an alarm type, send a GET ws/AlarmTemplate request to get a list of all available alarm templates.

```
PUT ws/Alarm
```

```
<Alarm>
    <almtId>12</almtId>   <!-- Missing Smart Energy DataPoint alarm -->
    <almName>Missing Smart Energy DataPoint</almName>
    <almDescription>Fires when devices have not reported SmartEnergy data
             within the specified time</almDescription>
    <almScopeConfig>
      <ScopingOptions>
        <Scope name="Group" value="/CUS001_ABC/" />
      </ScopingOptions>
    </almScopeConfig>
    <almRuleConfig>
      <Rules>
        <FireRule>
          <Variable name="uploadTimeUnit" value="hours" />
          <Variable name="clusterType" value="*" />
          <Variable name="readingTimeUnit" value="4" />
          <Variable name="attributeId" value="4" />
          <Variable name="uploadInterval" value="1" />
          <Variable name="clusterId" value="*" />
          <Variable name="endpointId" value="*" />
          <Variable name="readingInterval" value="10" />
       </FireRule>
       <ResetRule />
     </Rules>
   </almRuleConfig>
 </Alarm>
```

Example: Create a subscription usage alarm

The following sample shows how to create a subscription usage alarm that fires when Verizon cellular usage data exceeds 2 MB. The subscription usage alarm must specify the svcID along with a metric. Use the CustomerRatePlan web service to get a list of svcIDs.

Note In this example, the alarm template ID for the subscription usage alarm is 8 (almtId=8). To find the almtId for an alarm type, send a GET ws/AlarmTemplate request to get a list of all available alarm templates.

```
PUT ws/Alarm
```

```
   <Alarm>
      <almtId>8</almtId>   <!-- Subscription Usage alarm -->
      <almName>Verizon Cellular Usage</almName>
      <almDescription>Fires when verizon cellular usage data exceeds 2MB
</almDescription>
      <almScopeConfig>
        <ScopingOptions>
          <Scope name="Device" value="00000000-00000000-000000FF-FF000001" />
        </ScopingOptions>
      </almScopeConfig>
      <almRuleConfig>
        <Rules>
          <FireRule>
            <Variable name="unit" value="mb" />
            <Variable name="thresholdValue" value="2" />
            <Variable name="svcId" value="14" />
            <Variable name="metric" value="transferred" />
          </FireRule>
          <ResetRule />
        </Rules>
      </almRuleConfig>
   </Alarm>
```

AlarmStatus

Use the AlarmStatus web service to retrieve the current status of one or more alarms or to update the status of an alarm.

URI

```
http://<hostname>/ws/AlarmStatus
```

Formats

HTTP method	Format	Description
GET	/ws/AlarmStatus	Get statuses for all current alarms.
GET	/ws/AlarmStatus/{almId}	Get statuses for a specific alarm.
PUT	/ws/AlarmStatus/{almId}/{almsSourceEntityId}	Update the alarm status.

Elements

id

Unique identifier for the alarm status that consists of two elements:

ID Element	Description
almId	System-generated identifier for the alarm.
almsSourceEntityId	System-generated identifier for the entity associated with the alarm status, such as a device or task.

almsStatus

Current status of the alarm:

Status value	Description
0	Alarm is reset.

Status value	Description
1	Alarm is triggered.
2	Alarm is acknowledged.

almsTopic

Topic for the alarm.

cstId

Remote Manager identifier for the customer.

almsUpdateTime

Time at which the alarm status was last updated (ISO 8601 standard format).

almsPayload

Payload associated with the status change for the alarm in XML format. The payload can be any event object in the system that caused the status of the alarm to change. Typically, the payload is a web services object, such as a monitor or device core object.

Example: Get statuses for all alarms

The following sample request shows how to get a list of all alarm statuses for all alarms.

Request

```
GET ws/AlarmStatus
```

Response

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
   <resultTotalRows>4</resultTotalRows>
   <requestedStartRow>0</requestedStartRow>
   <resultSize>4</resultSize>
   <requestedSize>1000</requestedSize>
   <remainingSize>0</remainingSize>
   <AlarmStatus>
      <id>
         <almId>142</almId>   <!-- alarm 142 almId #1 -->
         <almsSourceEntityId>46911</almsSourceEntityId>
      </id>
      <almsStatus>2</almsStatus>
      <almsTopic>Alarm.System.Monitor.inactive</almsTopic>
      <cstId>2</cstId>
      <almsUpdateTime>2012-06-27T21:02:09.567Z</almsUpdateTime>
      <almsPayload>
         <Payload>
            <Message>Monitor disconnected: node left the cluster</Message>
            <Monitor>
               <monId>46911</monId>
               <cstId>2</cstId>
               <monLastConnect>2012-06-27T17:08:27.457Z</monLastConnect>

<monTopic>AlarmTemplate,Alarm,AlarmStatus,DeviceCore,XbeeCore</monTopic>
               <monTransportType>alarm</monTransportType>
               <monFormatType>xml</monFormatType>
               <monBatchSize>100</monBatchSize>
               <monCompression>none</monCompression>
               <monStatus>1</monStatus>
               <monBatchDuration>10</monBatchDuration>
            </Monitor>
         </Payload>
      </almsPayload>
   </AlarmStatus>
   <AlarmStatus>
      <id>
         <almId>142</almId>   <!-- alarm 142 almId #2 -->
         <almsSourceEntityId>Monitor:46911</almsSourceEntityId>
      </id>
      <almsStatus>0</almsStatus>
      <almsTopic>Alarm.System.Monitor.active</almsTopic>
      <cstId>2</cstId>
      <almsUpdateTime>2012-06-27T22:01:40.953Z</almsUpdateTime>
      <almsPayload>
         <Payload>
```

```
            <Message>Monitor connected</Message>
            <Monitor>
               <monId>46911</monId>
               <cstId>2</cstId>
               <monLastConnect>2012-06-27T21:39:50.833Z</monLastConnect>
               <monTopic>AlarmStatus,AlarmTemplate,Notification,Alarm</monTopic>
               <monTransportType>alarm</monTransportType>
               <monFormatType>xml</monFormatType>
               <monBatchSize>100</monBatchSize>
               <monCompression>none</monCompression>
               <monStatus>0</monStatus>
               <monBatchDuration>10</monBatchDuration>
            </Monitor>
         </Payload>
      </almsPayload>
   </AlarmStatus>
   <AlarmStatus>
      <id>
         <almId>151</almId>   <!-- alarm 151 almId #1 -->
         <almsSourceEntityId>00000000-00000000-00409DFF-
FF441634</almsSourceEntityId>
      </id>
      <almsStatus>1</almsStatus>
      <almsTopic>Alarm.DeviceOffline</almsTopic>
      <cstId>2</cstId>
      <almsUpdateTime>2012-07-02T15:25:57.387Z</almsUpdateTime>
      <almsPayload>
         <Payload>
            <DeviceCore>
               <id>
                  <devId>11116</devId>
                  <devVersion>0</devVersion>
               </id>
               <devRecordStartDate>2012-07-02T13:27:00.000Z</devRecordStartDate>
               <devMac>00:40:9D:44:16:34</devMac>
               <devConnectwareId>00000000-00000000-00409DFF-
FF441634</devConnectwareId>
               <cstId>2</cstId>
               <grpId>2</grpId>
               <devEffectiveStartDate>2012-07-
02T13:27:00.000Z</devEffectiveStartDate>
               <devTerminated>false</devTerminated>
               <dvVendorId>4261412867</dvVendorId>
               <dpDeviceType>CPX2e SE</dpDeviceType>
               <dpFirmwareLevel>50331744</dpFirmwareLevel>
               <dpFirmwareLevelDesc>3.0.0.96</dpFirmwareLevelDesc>
               <dpRestrictedStatus>0</dpRestrictedStatus>
               <dpLastKnownIp>10.9.16.17</dpLastKnownIp>
               <dpGlobalIp>66.77.174.126</dpGlobalIp>
               <dpConnectionStatus>0</dpConnectionStatus>
               <dpLastConnectTime>2012-07-02T13:26:35.627Z</dpLastConnectTime>
               <dpContact />
               <dpDescription />
               <dpLocation />
               <dpPanId>0xf02d</dpPanId>
               <xpExtAddr>00:13:A2:00:40:5C:0A:6A</xpExtAddr>
               <dpServerId />
               <dpZigbeeCapabilities>875</dpZigbeeCapabilities>
               <grpPath>/CUS0000001_Digi_International/</grpPath>
```

```
            </DeviceCore>
         </Payload>
      </almsPayload>
   </AlarmStatus>
   <AlarmStatus>
      <id>
         <almId>152</almId>   <!-- alarm 152 almId #1 -->
         <almsSourceEntityId>Monitor:47827</almsSourceEntityId>
      </id>
      <almsStatus>0</almsStatus>
      <almsTopic>Alarm.System.Monitor.active</almsTopic>
      <cstId>2</cstId>
      <almsUpdateTime>2012-07-02T02:10:57.130Z</almsUpdateTime>
      <almsPayload>
         <Payload>
            <Message>Monitor connected</Message>
            <Monitor>
               <monId>47827</monId>
               <cstId>2</cstId>
               <monLastConnect>2012-06-29T19:18:10.287Z</monLastConnect>

<monTopic>XbeeCore,DeviceCore,AlarmStatus,AlarmTemplate,Notification,Alarm</monTo
pic>
               <monTransportType>alarm</monTransportType>
               <monFormatType>xml</monFormatType>
               <monBatchSize>100</monBatchSize>
               <monCompression>none</monCompression>
               <monStatus>1</monStatus>
               <monBatchDuration>10</monBatchDuration>
            </Monitor>
         </Payload>
      </almsPayload>
   </AlarmStatus>
</result>
```

Example: Acknowledge a fired alarm

The following sample request shows how to acknowledge a fired alarm. The sample alarm ID is 3140, the almsSourceEntity for the alarm event is 00000000-00000000-00409DFF-FF53231C.

Request

```
PUT ws/AlarmStatus/3140/00000000-00000000-00409DFF-FF53231C/
```

```
<AlarmStatus>
   <almsStatus>2</almsStatus>
</AlarmStatus>
```

```
GET ws/AlarmStatus/3140/00000000-00000000-00409DFF-FF53231C/
```

Response

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
  <resultTotalRows>1</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>1</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <AlarmStatus>
    <id>
      <almId>3140</almId>
      <almsSourceEntityId>00000000-00000000-00409DFF-
FF53231C</almsSourceEntityId>
    </id>
    <almsStatus>2</almsStatus>
    <almsTopic>Alarm.DeviceOffline</almsTopic>
    <cstId>2</cstId>
    <almsUpdateTime>2014-07-07T22:06:26.193Z</almsUpdateTime>
    <almsPayload>
      <Payload>
        <Method>Manual</Method>
      </Payload>
    </almsPayload>
  </AlarmStatus>
</result>
```

Example: Reset a fired alarm

The following sample request uses the PUT method with the AlarmStatus web service to reset a fired alarm. The almID is 3140 and almsSourceEntity for the alarm event is 00000000-00000000-00409DFF-FF53231C.

Request

```
PUT ws/AlarmStatus/3140/00000000-00000000-00409DFF-FF53231C/
```

```
<AlarmStatus>
   <almsStatus>0</almsStatus>
</AlarmStatus>
```

```
GET ws/AlarmStatus/3140/00000000-00000000-00409DFF-FF53231C/
```

Response

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
  <resultTotalRows>1</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>1</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <AlarmStatus>
    <id>
      <almId>3140</almId>
      <almsSourceEntityId>00000000-00000000-00409DFF-
FF53231C</almsSourceEntityId>
    </id>
    <almsStatus>0</almsStatus>
    <almsTopic>Alarm.DeviceOffline</almsTopic>
    <cstId>2</cstId>
    <almsUpdateTime>2014-07-07T22:06:26.193Z</almsUpdateTime>
    <almsPayload>
      <Payload>
        <Method>Manual</Method>
      </Payload>
    </almsPayload>
  </AlarmStatus>
</result>
```

AlarmStatusHistory

Use the AlarmStatusHistory web service to retrieve a record of alarm statuses over time.

URI

```
http://<hostname>/ws/AlarmStatusHistory
```

Formats

HTTP method	Format	Description
GET	/ws/AlarmStatusHistory	Get a list of all alarm statuses over time.
GET	/ws/AlarmStatusHistory/{almId}	Get a list of alarm statuses over time for a specific alarm.

Elements

id

Unique identifier for the alarm status that consists of two elements:

ID Element	Description
almId	System-generated identifier for the alarm.
almsSourceEntityId	System-generated identifier for the entity associated with the alarm status, such as a device or task.

almsStatus

Current status of the alarm:

Status value	Description
0	Alarm is reset.
1	Alarm is triggered.
2	Alarm is acknowledged.

almsTopic

Topic for the alarm.

cstId

Remote Manager identifier for the customer.

almsUpdateTime

Time at which the alarm status was last updated (ISO 8601 standard format).

almsPayload

Payload associated with the status change for the alarm in XML format. The payload can be any event object in the system that caused the status of the alarm to change. Typically, the payload is a web services object, such as a monitor or device core object.

Query parameters

size

Number of resources to return for a GET request. Allowable value is a number from 1 to 1000. The default is 1000.

pageCursor

Page cursor that was returned from a previous request that can be used to retrieve the next page of data.

startTime

Start time (inclusive) for the status history you want to retrieve.

endTime

End time (exclusive) for the status history you want to retrieve.

order

Sort order for the retrieved data: **asc** for ascending or **desc** for descending.

Example: Get a list of all alarm statuses over time

The following sample request shows how to get all statuses for all configured alarms over time.

Request

```
GET ws/AlarmStatusHistory
```

Response (abbreviated)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
   <resultSize>580</resultSize>
   <requestedSize>1000</requestedSize>
   <pageCursor>5a0319a4-7c95-11e4-a62c-fa163e4e63b3</pageCursor>
   <requestedStartTime>-1</requestedStartTime>
   <requestedEndTime>-1</requestedEndTime>
   <AlarmStatusHistory>
      <id>
         <almId>9219</almId>
         <almsSourceEntityId>00000000-00000000-BBCCDDFF-
FF004000</almsSourceEntityId>
      </id>
      <almsStatus>0</almsStatus>
      <almsTopic>Alarm.DeviceOffline</almsTopic>
      <cstId>2</cstId>
      <almsUpdateTime>2014-11-10T23:48:42.594Z</almsUpdateTime>
      <almsPayload>
         <Payload>
            <DeviceCore>
               <id>
                  <devId>317792</devId>
                  <devVersion>0</devVersion>
               </id>
               <devRecordStartDate>2014-08-13T17:41:00.000Z</devRecordStartDate>
               <devMac>BB:CC:DD:00:40:00</devMac>
               <devConnectwareId>00000000-00000000-BBCCDDFF-
FF004000</devConnectwareId>
               <cstId>2</cstId>
               <grpId>20686</grpId>
               <devEffectiveStartDate>2014-08-
13T17:41:00.000Z</devEffectiveStartDate>
               <devTerminated>false</devTerminated>
               <dvVendorId>50331650</dvVendorId>
               <dpDeviceType>Joshs Device</dpDeviceType>
               <dpFirmwareLevel>16777216</dpFirmwareLevel>
               <dpFirmwareLevelDesc>1.0.0.0</dpFirmwareLevelDesc>
               <dpRestrictedStatus>0</dpRestrictedStatus>
               <dpLastKnownIp>199.17.162.22</dpLastKnownIp>
               <dpGlobalIp>199.17.162.22</dpGlobalIp>
               <dpConnectionStatus>1</dpConnectionStatus>
               <dpLastConnectTime>2014-11-10T23:48:42.394Z</dpLastConnectTime>
               <dpContact/>
               <dpDescription/>
               <dpLocation/>
               <dpMapLat>44.932017</dpMapLat>
```

```
               <dpMapLong>-93461594000000.000000</dpMapLong>
               <dpServerId>ClientID[5]</dpServerId>
               <dpZigbeeCapabilities>0</dpZigbeeCapabilities>
               <dpCapabilities>68178</dpCapabilities>
               <grpPath>/CUS0000002_Systems_Assurance/WSU/</grpPath>
               <dpLastDisconnectTime>2014-11-
03T22:46:03.460Z</dpLastDisconnectTime>
               <dpLastUpdateTime>2014-10-22T15:52:44.247Z</dpLastUpdateTime>
               <dpHealthStatus>-1</dpHealthStatus>
            </DeviceCore>
         </Payload>
      </almsPayload>
      <almsSeverity>1</almsSeverity>
   </AlarmStatusHistory>
   <AlarmStatusHistory>
      <id>
         <almId>9219</almId>
         <almsSourceEntityId>00000000-00000000-BBCCDDFF-
FF004000</almsSourceEntityId>
      </id>
      <almsStatus>1</almsStatus>
      <almsTopic>Alarm.DeviceOffline</almsTopic>
      <cstId>2</cstId>
      <almsUpdateTime>2014-11-11T03:42:29.477Z</almsUpdateTime>
      <almsPayload>
         <Payload>
            <DeviceCore>
               <id>
                  <devId>317792</devId>
                  <devVersion>0</devVersion>
               </id>
               <devRecordStartDate>2014-08-13T17:41:00.000Z</devRecordStartDate>
               <devMac>BB:CC:DD:00:40:00</devMac>
               <devConnectwareId>00000000-00000000-BBCCDDFF-
FF004000</devConnectwareId>
               <cstId>2</cstId>
               <grpId>20686</grpId>
               <devEffectiveStartDate>2014-08-
13T17:41:00.000Z</devEffectiveStartDate>
               <devTerminated>false</devTerminated>
               <dvVendorId>50331650</dvVendorId>
               <dpDeviceType>Joshs Device</dpDeviceType>
               <dpFirmwareLevel>16777216</dpFirmwareLevel>
               <dpFirmwareLevelDesc>1.0.0.0</dpFirmwareLevelDesc>
               <dpRestrictedStatus>0</dpRestrictedStatus>
               <dpLastKnownIp>199.17.162.22</dpLastKnownIp>
               <dpGlobalIp>199.17.162.22</dpGlobalIp>
               <dpConnectionStatus>0</dpConnectionStatus>
               <dpLastConnectTime>2014-11-10T23:48:42.393Z</dpLastConnectTime>
               <dpContact/>
               <dpDescription/>
               <dpLocation/>
               <dpMapLat>44.932017</dpMapLat>
               <dpMapLong>-93461594000000.000000</dpMapLong>
               <dpServerId/>
               <dpZigbeeCapabilities>0</dpZigbeeCapabilities>
               <dpCapabilities>68178</dpCapabilities>
               <grpPath>/CUS0000002_Systems_Assurance/WSU/</grpPath>
               <dpLastDisconnectTime>2014-11-
```

```
11T03:37:29.368Z</dpLastDisconnectTime>
               <dpLastUpdateTime>2014-10-22T15:52:44.247Z</dpLastUpdateTime>
               <dpHealthStatus>-1</dpHealthStatus>
            </DeviceCore>
         </Payload>
      </almsPayload>
      <almsSeverity>1</almsSeverity>
   </AlarmStatusHistory>
```

Example: Get alarm status history for a specific alarm

The following sample request shows how to retrieve alarm status history for alarm ID 9219 with source entity ID 00000000-00000000-BBCCDDFF-FF004000.

Request

```
GET ws/AlarmStatusHistory/9219/00000000-00000000-BBCCDDFF-FF004000/
```

Response (abbreviated)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
   <resultSize>29</resultSize>
   <requestedSize>1000</requestedSize>
   <pageCursor>3ab1d732-7c95-11e4-a62c-fa163e4e63b3</pageCursor>
   <requestedStartTime>-1</requestedStartTime>
   <requestedEndTime>-1</requestedEndTime>
   <AlarmStatusHistory>
      <id>
         <almId>9219</almId>
         <almsSourceEntityId>00000000-00000000-BBCCDDFF-
FF004000</almsSourceEntityId>
      </id>
      <almsStatus>0</almsStatus>
      <almsTopic>Alarm.DeviceOffline</almsTopic>
      <cstId>2</cstId>
      <almsUpdateTime>2014-11-10T23:48:42.594Z</almsUpdateTime>
      <almsPayload>
         <Payload>
            <DeviceCore>
               <id>
                  <devId>317792</devId>
                  <devVersion>0</devVersion>
               </id>
               <devRecordStartDate>2014-08-13T17:41:00.000Z</devRecordStartDate>
               <devMac>BB:CC:DD:00:40:00</devMac>
               <devConnectwareId>00000000-00000000-BBCCDDFF-
FF004000</devConnectwareId>
               <cstId>2</cstId>
               <grpId>20686</grpId>
               <devEffectiveStartDate>2014-08-
13T17:41:00.000Z</devEffectiveStartDate>
               <devTerminated>false</devTerminated>
               <dvVendorId>50331650</dvVendorId>
               <dpDeviceType>Joshs Device</dpDeviceType>
               <dpFirmwareLevel>16777216</dpFirmwareLevel>
               <dpFirmwareLevelDesc>1.0.0.0</dpFirmwareLevelDesc>
               <dpRestrictedStatus>0</dpRestrictedStatus>
               <dpLastKnownIp>199.17.162.22</dpLastKnownIp>
               <dpGlobalIp>199.17.162.22</dpGlobalIp>
               <dpConnectionStatus>1</dpConnectionStatus>
               <dpLastConnectTime>2014-11-10T23:48:42.394Z</dpLastConnectTime>
               <dpContact/>
               <dpDescription/>
               <dpLocation/>
```

```
               <dpMapLat>44.932017</dpMapLat>
               <dpMapLong>-93461594000000.000000</dpMapLong>
               <dpServerId>ClientID[5]</dpServerId>
               <dpZigbeeCapabilities>0</dpZigbeeCapabilities>
               <dpCapabilities>68178</dpCapabilities>
               <grpPath>/CUS0000002_Systems_Assurance/WSU/</grpPath>
               <dpLastDisconnectTime>2014-11-
03T22:46:03.460Z</dpLastDisconnectTime>
               <dpLastUpdateTime>2014-10-22T15:52:44.247Z</dpLastUpdateTime>
               <dpHealthStatus>-1</dpHealthStatus>
            </DeviceCore>
         </Payload>
      </almsPayload>
      <almsSeverity>1</almsSeverity>
   </AlarmStatusHistory>
   <AlarmStatusHistory>
      <id>
         <almId>9219</almId>
         <almsSourceEntityId>00000000-00000000-BBCCDDFF-
FF004000</almsSourceEntityId>
      </id>
      <almsStatus>1</almsStatus>
      <almsTopic>Alarm.DeviceOffline</almsTopic>
      <cstId>2</cstId>
      <almsUpdateTime>2014-11-11T03:42:29.477Z</almsUpdateTime>
      <almsPayload>
         <Payload>
            <DeviceCore>
               <id>
                  <devId>317792</devId>
                  <devVersion>0</devVersion>
               </id>
               <devRecordStartDate>2014-08-13T17:41:00.000Z</devRecordStartDate>
               <devMac>BB:CC:DD:00:40:00</devMac>
               <devConnectwareId>00000000-00000000-BBCCDDFF-
FF004000</devConnectwareId>
               <cstId>2</cstId>
               <grpId>20686</grpId>
               <devEffectiveStartDate>2014-08-
13T17:41:00.000Z</devEffectiveStartDate>
               <devTerminated>false</devTerminated>
               <dvVendorId>50331650</dvVendorId>
               <dpDeviceType>Joshs Device</dpDeviceType>
               <dpFirmwareLevel>16777216</dpFirmwareLevel>
               <dpFirmwareLevelDesc>1.0.0.0</dpFirmwareLevelDesc>
               <dpRestrictedStatus>0</dpRestrictedStatus>
               <dpLastKnownIp>199.17.162.22</dpLastKnownIp>
               <dpGlobalIp>199.17.162.22</dpGlobalIp>
               <dpConnectionStatus>0</dpConnectionStatus>
               <dpLastConnectTime>2014-11-10T23:48:42.393Z</dpLastConnectTime>
               <dpContact/>
               <dpDescription/>
               <dpLocation/>
               <dpMapLat>44.932017</dpMapLat>
               <dpMapLong>-93461594000000.000000</dpMapLong>
               <dpServerId/>
               <dpZigbeeCapabilities>0</dpZigbeeCapabilities>
               <dpCapabilities>68178</dpCapabilities>
               <grpPath>/CUS0000002_Systems_Assurance/WSU/</grpPath>
```

```
               <dpLastDisconnectTime>2014-11-
11T03:37:29.368Z</dpLastDisconnectTime>
               <dpLastUpdateTime>2014-10-22T15:52:44.247Z</dpLastUpdateTime>
               <dpHealthStatus>-1</dpHealthStatus>
            </DeviceCore>
         </Payload>
      </almsPayload>
      <almsSeverity>1</almsSeverity>
   </AlarmStatusHistory>
   <AlarmStatusHistory>
      <id>
         <almId>9219</almId>
         <almsSourceEntityId>00000000-00000000-BBCCDDFF-
FF004000</almsSourceEntityId>
      </id>
      <almsStatus>0</almsStatus>
      <almsTopic>Alarm.DeviceOffline</almsTopic>
      <cstId>2</cstId>
      <almsUpdateTime>2014-11-11T18:26:08.073Z</almsUpdateTime>
      <almsPayload>
         <Payload>
            <DeviceCore>
               <id>
                  <devId>317792</devId>
                  <devVersion>0</devVersion>
               </id>
               <devRecordStartDate>2014-08-13T17:41:00.000Z</devRecordStartDate>
               <devMac>BB:CC:DD:00:40:00</devMac>
               <devConnectwareId>00000000-00000000-BBCCDDFF-
FF004000</devConnectwareId>
               <cstId>2</cstId>
               <grpId>20686</grpId>
               <devEffectiveStartDate>2014-08-
13T17:41:00.000Z</devEffectiveStartDate>
               <devTerminated>false</devTerminated>
               <dvVendorId>50331650</dvVendorId>
               <dpDeviceType>Joshs Device</dpDeviceType>
               <dpFirmwareLevel>16777216</dpFirmwareLevel>
               <dpFirmwareLevelDesc>1.0.0.0</dpFirmwareLevelDesc>
               <dpRestrictedStatus>0</dpRestrictedStatus>
               <dpLastKnownIp>199.17.162.22</dpLastKnownIp>
               <dpGlobalIp>199.17.162.22</dpGlobalIp>
               <dpConnectionStatus>1</dpConnectionStatus>
               <dpLastConnectTime>2014-11-11T18:26:07.903Z</dpLastConnectTime>
               <dpContact/>
               <dpDescription/>
               <dpLocation/>
               <dpMapLat>44.932017</dpMapLat>
               <dpMapLong>-93461594000000.000000</dpMapLong>
               <dpServerId>ClientID[4]</dpServerId>
               <dpZigbeeCapabilities>0</dpZigbeeCapabilities>
               <dpCapabilities>68178</dpCapabilities>
               <grpPath>/CUS0000002_Systems_Assurance/WSU/</grpPath>
               <dpLastDisconnectTime>2014-11-
11T03:37:29.367Z</dpLastDisconnectTime>
               <dpLastUpdateTime>2014-10-22T15:52:44.247Z</dpLastUpdateTime>
               <dpHealthStatus>-1</dpHealthStatus>
            </DeviceCore>
         </Payload>
```

```
      </almsPayload>
      <almsSeverity>1</almsSeverity>
   </AlarmStatusHistory>
   <AlarmStatusHistory>
      <id>
         <almId>9219</almId>
         <almsSourceEntityId>00000000-00000000-BBCCDDFF-
FF004000</almsSourceEntityId>
      </id>
      <almsStatus>1</almsStatus>
      <almsTopic>Alarm.DeviceOffline</almsTopic>
      <cstId>2</cstId>
      <almsUpdateTime>2014-11-11T19:04:15.977Z</almsUpdateTime>
      <almsPayload>
         <Payload>
            <DeviceCore>
               <id>
                  <devId>317792</devId>
                  <devVersion>0</devVersion>
               </id>
               <devRecordStartDate>2014-08-13T17:41:00.000Z</devRecordStartDate>
               <devMac>BB:CC:DD:00:40:00</devMac>
               <devConnectwareId>00000000-00000000-BBCCDDFF-
FF004000</devConnectwareId>
               <cstId>2</cstId>
               <grpId>20686</grpId>
               <devEffectiveStartDate>2014-08-
13T17:41:00.000Z</devEffectiveStartDate>
               <devTerminated>false</devTerminated>
               <dvVendorId>50331650</dvVendorId>
               <dpDeviceType>Joshs Device</dpDeviceType>
               <dpFirmwareLevel>16777216</dpFirmwareLevel>
               <dpFirmwareLevelDesc>1.0.0.0</dpFirmwareLevelDesc>
               <dpRestrictedStatus>0</dpRestrictedStatus>
               <dpLastKnownIp>199.17.162.22</dpLastKnownIp>
               <dpGlobalIp>199.17.162.22</dpGlobalIp>
               <dpConnectionStatus>0</dpConnectionStatus>
               <dpLastConnectTime>2014-11-11T18:26:07.903Z</dpLastConnectTime>
               <dpContact/>
               <dpDescription/>
               <dpLocation/>
               <dpMapLat>44.932017</dpMapLat>
               <dpMapLong>-93461594000000.000000</dpMapLong>
               <dpServerId/>
               <dpZigbeeCapabilities>0</dpZigbeeCapabilities>
               <dpCapabilities>68178</dpCapabilities>
               <grpPath>/CUS0000002_Systems_Assurance/WSU/</grpPath>
               <dpLastDisconnectTime>2014-11-
11T18:59:15.868Z</dpLastDisconnectTime>
               <dpLastUpdateTime>2014-10-22T15:52:44.247Z</dpLastUpdateTime>
               <dpHealthStatus>-1</dpHealthStatus>
            </DeviceCore>
         </Payload>
      </almsPayload>
      <almsSeverity>1</almsSeverity>
   </AlarmStatusHistory>
   <AlarmStatusHistory>
      <id>
         <almId>9219</almId>
```

```
         <almsSourceEntityId>00000000-00000000-BBCCDDFF-
FF004000</almsSourceEntityId>
      </id>
      <almsStatus>0</almsStatus>
      <almsTopic>Alarm.DeviceOffline</almsTopic>
      <cstId>2</cstId>
      <almsUpdateTime>2014-11-11T23:43:52.707Z</almsUpdateTime>
      <almsPayload>
         <Payload>
            <DeviceCore>
               <id>
                  <devId>317792</devId>
                  <devVersion>0</devVersion>
               </id>
               <devRecordStartDate>2014-08-13T17:41:00.000Z</devRecordStartDate>
               <devMac>BB:CC:DD:00:40:00</devMac>
               <devConnectwareId>00000000-00000000-BBCCDDFF-
FF004000</devConnectwareId>
               <cstId>2</cstId>
               <grpId>20686</grpId>
               <devEffectiveStartDate>2014-08-
13T17:41:00.000Z</devEffectiveStartDate>
               <devTerminated>false</devTerminated>
               <dvVendorId>50331650</dvVendorId>
               <dpDeviceType>Joshs Device</dpDeviceType>
               <dpFirmwareLevel>16777216</dpFirmwareLevel>
               <dpFirmwareLevelDesc>1.0.0.0</dpFirmwareLevelDesc>
               <dpRestrictedStatus>0</dpRestrictedStatus>
               <dpLastKnownIp>199.17.162.22</dpLastKnownIp>
               <dpGlobalIp>199.17.162.22</dpGlobalIp>
               <dpConnectionStatus>1</dpConnectionStatus>
               <dpLastConnectTime>2014-11-11T23:43:52.632Z</dpLastConnectTime>
               <dpContact/>
               <dpDescription/>
               <dpLocation/>
               <dpMapLat>44.932017</dpMapLat>
               <dpMapLong>-93461594000000.000000</dpMapLong>
               <dpServerId>ClientID[4]</dpServerId>
               <dpZigbeeCapabilities>0</dpZigbeeCapabilities>
               <dpCapabilities>68178</dpCapabilities>
               <grpPath>/CUS0000002_Systems_Assurance/WSU/</grpPath>
               <dpLastDisconnectTime>2014-11-
11T18:59:15.867Z</dpLastDisconnectTime>
               <dpLastUpdateTime>2014-10-22T15:52:44.247Z</dpLastUpdateTime>
               <dpHealthStatus>-1</dpHealthStatus>
            </DeviceCore>
         </Payload>
      </almsPayload>
      <almsSeverity>1</almsSeverity>
   </AlarmStatusHistory>
   <AlarmStatusHistory>
      <id>
         <almId>9219</almId>
         <almsSourceEntityId>00000000-00000000-BBCCDDFF-
FF004000</almsSourceEntityId>
      </id>
      <almsStatus>1</almsStatus>
      <almsTopic>Alarm.DeviceOffline</almsTopic>
      <cstId>2</cstId>
```

```
      <almsUpdateTime>2014-11-11T23:56:33.677Z</almsUpdateTime>
      <almsPayload>
         <Payload>
            <DeviceCore>
               <id>
                  <devId>317792</devId>
                  <devVersion>0</devVersion>
               </id>
               <devRecordStartDate>2014-08-13T17:41:00.000Z</devRecordStartDate>
               <devMac>BB:CC:DD:00:40:00</devMac>
               <devConnectwareId>00000000-00000000-BBCCDDFF-
FF004000</devConnectwareId>
               <cstId>2</cstId>
               <grpId>20686</grpId>
               <devEffectiveStartDate>2014-08-
13T17:41:00.000Z</devEffectiveStartDate>
               <devTerminated>false</devTerminated>
               <dvVendorId>50331650</dvVendorId>
               <dpDeviceType>Joshs Device</dpDeviceType>
               <dpFirmwareLevel>16777216</dpFirmwareLevel>
               <dpFirmwareLevelDesc>1.0.0.0</dpFirmwareLevelDesc>
               <dpRestrictedStatus>0</dpRestrictedStatus>
               <dpLastKnownIp>199.17.162.22</dpLastKnownIp>
               <dpGlobalIp>199.17.162.22</dpGlobalIp>
               <dpConnectionStatus>0</dpConnectionStatus>
               <dpLastConnectTime>2014-11-11T23:43:52.633Z</dpLastConnectTime>
               <dpContact/>
               <dpDescription/>
               <dpLocation/>
               <dpMapLat>44.932017</dpMapLat>
               <dpMapLong>-93461594000000.000000</dpMapLong>
               <dpServerId/>
               <dpZigbeeCapabilities>0</dpZigbeeCapabilities>
               <dpCapabilities>68178</dpCapabilities>
               <grpPath>/CUS0000002_Systems_Assurance/WSU/</grpPath>
               <dpLastDisconnectTime>2014-11-
11T23:51:33.617Z</dpLastDisconnectTime>
               <dpLastUpdateTime>2014-10-22T15:52:44.247Z</dpLastUpdateTime>
               <dpHealthStatus>-1</dpHealthStatus>
            </DeviceCore>
         </Payload>
      </almsPayload>
      <almsSeverity>1</almsSeverity>
   </AlarmStatusHistory>
```

AlarmTemplate

Use the AlarmTemplate web service to retrieve information about available alarm types within your Remote Manager host.

URI

```
http://<hostname>/ws/AlarmTemplate
```

Formats

HTTP method	Format	Description
GET	/ws/AlarmTemplate	Get a list of all available alarm templates.

Elements

almtID

System-generated identifier for an alarm template. To get a list of available alarm template, use the AlarmTemplate web service.

almtName

Name of the alarm template.

almtDescription

Description of the alarm template.

grpId

Remote Manager identifier for the customer group.

almtTopic

Topic for the alarm template.

almtScopeOptions

Specifies the resource scope for the alarm template. Scope options include:

Scope	Description
Group	Applies the alarm to the specified group indicated by the full group path.

Scope	Description
Device	Applies the alarm to the specified device. For example, 00000000-00000000-00000000-00000000.
XbeeNode	Applies the alarm to the specified XbeeNode, expressed as the XbeeNode extended address. For example, 00:13:A2:00:00:00:00:00.
Resource	Applies the alarm to a data stream path or pattern. You can use the wildcard charater asterisk (*) to match any element in the data stream path. For example, dia/channel/*/lth/temp matches all the lth temperature reading for all devices.
Global	Applies the alarm at the customer level to monitor all instances of an entity. For example, you can create an alarm to monitor the total number of web services calls for your account. This option is available for subscription usage alarms only.

See almtScopeOptions for the required XML structure for almtScopeOptions.

almtRules

Specifies the rule configurations for the alarm template:

Rule configuration	Description
FireRule	A list of variables that specify the condition for firing the alarm.
ResetRule	A list of variables that specify the condition for resetting the alarm.

See almtRules for the required XML structure for almtRules.

See Alarm template types for a list of available fire and reset variables for each alarm template type.

Alarm template types

The following table lists and describes all available alarm template types.

Alarm template type	Description	Scoping Options	Fire variables	Reset variables
Device offline	Detects when a device disconnects from Remote Manager and fails to reconnect withint the specified time.	Device Group	reconnectWindowDuration	none
XBeeNode offline	Detects when an XBee node disconnects from Remote Manager and fails to reconnect within the specified time.	Device Group XBeeNode	reconnectWindowDuration	none
Device excessive disconnects	Detects devices with an excessive number of disconnects.	Device Group	disconnectCount disconnectWindow	reconnectWindow
XBeeNode excessive deactivations	Detects XBee nodes with an excessive number of deactivations.	Device Group XBeeNode	deactivationCount deactivationWindow	activationWindow
DIA channel data point condition watch	Detects when the specified DIA channel conditions are met.	Device Group	instanceName channelName type operator thresholdValue timeout timeUnit	instanceName channelName type operator thresholdValue timeout timeUnit
Smart energy data point condition match	Detects when the specified DIA channel conditions are met.	Device Group XBeeNode	endpointID clusterType clusterID attributeID type operator thresholdValue timeout timeUnit	endpointID clusterType clusterID attributeID type operator thresholdValue timeout timeUnit

Alarm template type	Description	Scoping Options	Fire variables	Reset variables
Data point condition match	Detects when the specified data point usage conditions are met.	Resource	type operator thresholdValue timeout timeUnit	type operator thresholdValue timeout timeUnit
Subscription usage	Detects when subscription usage exceeds a specified threshold.	Device Group Global	svcID thresholdValue metric unit	none
Missing data point	Detects when one or more data points are not reported on within the specified time.	Resource	uploadInterval uploadTimeUnit readingInterval readingTimeUnit	none
Missing DIA channel data point	Detects when devices haven not reported DIA channel data within the specified time. **Note** The alarm will not be registered until the first data point is sent after the alarm is created or edited.	Device Group	instanceName channelName uploadInterval uploadTimeUnit readingInterval readingTimeUnit	none
Missing smart energy data point	Detects when devices have not reported smart energy data within the specifed time. **Note** The alarm will not be registered until the first data point is sent after the alarm is created or edited.	Device Group XBeeNode	endpointID clusterType clusterID attributedID uploadInterval uploadTimeUnit readingInterval readingTimeUnit	none

almtScopeOptions

Use the following XML structure for almtScopeOptions.

```
<almScopeConfig>
    <ScopingOptions>
        <Scope name="Resource" value="Weather/USA/*/Minneapolis"/>
    </ScopingOptions>
</almScopeConfig>
```

almtRules

Use the following structure for almtRules.

```
<almRuleConfig>
    <Rules>
       <FireRule>
         <Variable name="variableName" value="value"/>
         ...
         </FireRule>
         <ResetRule>
         <Variable name="variableName" value="value"/>
         ...
         </ResetRule>
    </Rules>
</almRuleConfig>
```

Example: List all alarm templates

The following sample request retrieves a list of all alarm templates for your Remote Manager host.

Request

```
GET ws/AlarmTemplate
```

Response

Header

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
   <resultTotalRows>11</resultTotalRows>
   <requestedStartRow>0</requestedStartRow>
   <resultSize>11</resultSize>
   <requestedSize>1000</requestedSize>
   <remainingSize>0</remainingSize>
```

Alarm template ID 2: Device offline

```
  <AlarmTemplate>
    <almtId>2</almtId>
    <almtName>Device Offline</almtName>
    <almtDescription>Detects when a device disconnects from Remote Manager and
fails to reconnected</almtDescription>
    <grpId>1</grpId>
    <almtTopic>Alarm.DeviceOffline</almtTopic>
    <almtScopeOptions>
      <ScopingOptions>
        <Scope name="Group"/>
        <Scope name="Device"/>
      </ScopingOptions>
    </almtScopeOptions>
    <almtRules>
      <Rules>
        <FireRule name="fireRule1">
          <Variable name="reconnectWindowDuration" type="int"/>
        </FireRule>
        <ResetRule name="resetRule1">
        </ResetRule>
      </Rules>
    </almtRules>
    <almtResourceList>DeviceCore,AlarmStatus</almtResourceList>
  </AlarmTemplate>
```

Alarm template ID 3: XBeeNode offline

```
  <AlarmTemplate>
    <almtId>3</almtId>
    <almtName>XBeeNode Offline</almtName>
    <almtDescription>Detects when an XBee Node disconnects from Remote Manager
and fails to reconnect</almtDescription>
    <grpId>1</grpId>
```

```
    <almtTopic>Alarm.XBeeNodeOffline</almtTopic>
    <almtScopeOptions>
      <ScopingOptions>
        <Scope name="Group"/>
        <Scope name="Device"/>
        <Scope name="XbeeNode"/>
      </ScopingOptions>
    </almtScopeOptions>
    <almtRules>
      <Rules>
        <FireRule name="fireRule1">
          <Variable name="reconnectWindowDuration" type="int"/>
        </FireRule>
        <ResetRule name="resetRule1">
        </ResetRule>
      </Rules>
    </almtRules>
    <almtResourceList>XbeeCore,AlarmStatus</almtResourceList>
  </AlarmTemplate>
```

Alarm template ID 4: Device excessive disconnects

```
  <AlarmTemplate>
    <almtId>4</almtId>
    <almtName>Device Excessive Disconnects</almtName>
    <almtDescription>Detects devices with an excessive number of
disconnects.</almtDescription>
    <grpId>1</grpId>
    <almtTopic>Alarm.DeviceExcessiveDisconnect</almtTopic>
    <almtScopeOptions>
      <ScopingOptions>
        <Scope name="Group"/>
        <Scope name="Device"/>
      </ScopingOptions>
    </almtScopeOptions>
    <almtRules>
      <Rules>
        <FireRule name="fireRule1">
          <Variable name="disconnectCount" type="int"/>
          <Variable name="disconnectWindow" type="int"/>
        </FireRule>
        <ResetRule name="resetRule1">
          <Variable name="reconnectWindow" type="int"/>
        </ResetRule>
      </Rules>
    </almtRules>
    <almtResourceList>DeviceCore,AlarmStatus</almtResourceList>
  </AlarmTemplate>
```

Alarm template ID 5: XBeeNode excessive deactivations

```
  <AlarmTemplate>
    <almtId>5</almtId>
    <almtName>XBeeNode Excessive Deactivations</almtName>
    <almtDescription>Detects XBeeNodes with an excessive number of
deactivations.</almtDescription>
    <grpId>1</grpId>
    <almtTopic>Alarm.XBeeNodeExcessiveDeactivation</almtTopic>
```

```
    <almtScopeOptions>
      <ScopingOptions>
        <Scope name="Group"/>
        <Scope name="Device"/>
        <Scope name="XbeeNode"/>
      </ScopingOptions>
    </almtScopeOptions>
    <almtRules>
      <Rules>
        <FireRule name="fireRule1">
          <Variable name="deactivationCount" type="int"/>
          <Variable name="deactivationWindow" type="int"/>
        </FireRule>
        <ResetRule name="resetRule1">
          <Variable name="activationWindow" type="int"/>
        </ResetRule>
      </Rules>
    </almtRules>
    <almtResourceList>XbeeCore,AlarmStatus</almtResourceList>
  </AlarmTemplate>
```

Alarm template ID 6: DIA channel data point condition match

```
  <AlarmTemplate>
    <almtId>6</almtId>
    <almtName>Dia channel data point condition match</almtName>
    <almtDescription>Detects dia channel condition</almtDescription>
    <grpId>1</grpId>
    <almtTopic>Alarm.DiaChannelDataPoint</almtTopic>
    <almtScopeOptions>
      <ScopingOptions>
        <Scope name="Group"/>
        <Scope name="Device"/>
      </ScopingOptions>
    </almtScopeOptions>
    <almtRules>
      <Rules>
        <FireRule name="fireRule1">
          <Variable name="instanceName" type="string"/>
          <Variable name="channelName" type="string"/>
          <Variable name="type" type="enum">
            <Value desc="Numeric" value="numeric"/>
            <Value desc="String" value="string"/>
          </Variable>
          <Variable name="operator" type="enum">
            <Value desc="&gt;" value="&gt;"/>
            <Value desc="&gt;=" value="&gt;="/>
            <Value desc="&lt;" value="&lt;"/>
            <Value desc="&lt;=" value="&lt;="/>
            <Value desc="=" value="="/>
            <Value desc="!=" value="&lt;&gt;"/>
          </Variable>
          <Variable name="thresholdValue" type="string"/>
          <Variable name="timeout" type="int"/>
          <Variable name="timeUnit" type="enum">
            <Value desc="Seconds" value="seconds"/>
            <Value desc="Minutes" value="minutes"/>
            <Value desc="Hours" value="hours"/>
```

```
          </Variable>
        </FireRule>
        <ResetRule name="resetRule1">
          <Variable name="instanceName" type="string"/>
          <Variable name="channelName" type="string"/>
          <Variable name="type" type="enum">
            <Value desc="Numeric" value="numeric"/>
            <Value desc="String" value="string"/>
          </Variable>
          <Variable name="operator" type="enum">
            <Value desc="&gt;" value="&gt;"/>
            <Value desc="&gt;=" value="&gt;="/>
            <Value desc="&lt;" value="&lt;"/>
            <Value desc="&lt;=" value="&lt;="/>
            <Value desc="=" value="="/>
            <Value desc="!=" value="&lt;&gt;"/>
          </Variable>
          <Variable name="thresholdValue" type="string"/>
          <Variable name="timeout" type="int"/>
          <Variable name="timeUnit" type="enum">
            <Value desc="Seconds" value="seconds"/>
            <Value desc="Minutes" value="minutes"/>
            <Value desc="Hours" value="hours"/>
          </Variable>
        </ResetRule>
      </Rules>
    </almtRules>
    <almtResourceList>DataPoint,AlarmStatus</almtResourceList>
  </AlarmTemplate>
```

Alarm template ID 7: Smart energy data point condition match

```
  <AlarmTemplate>
    <almtId>7</almtId>
    <almtName>Smart energy data point condition match</almtName>
    <almtDescription>Detects smart energy data point condition</almtDescription>
    <grpId>1</grpId>
    <almtTopic>Alarm.XbeeAttributeDataPoint</almtTopic>
    <almtScopeOptions>
      <ScopingOptions>
        <Scope name="Group"/>
        <Scope name="Device"/>
        <Scope name="XbeeNode"/>
      </ScopingOptions>
    </almtScopeOptions>
    <almtRules>
      <Rules>
        <FireRule name="fireRule1">
          <Variable name="endpointId" type="string"/>
          <Variable name="clusterType" type="string"/>
          <Variable name="clusterId" type="string"/>
          <Variable name="attributeId" type="string"/>
          <Variable name="type" type="enum">
            <Value desc="Numeric" value="numeric"/>
            <Value desc="String" value="string"/>
          </Variable>
          <Variable name="operator" type="enum">
            <Value desc="&gt;" value="&gt;"/>
```

```
            <Value desc="&gt;=" value="&gt;="/>
            <Value desc="&lt;" value="&lt;"/>
            <Value desc="&lt;=" value="&lt;="/>
            <Value desc="=" value="="/>
            <Value desc="!=" value="&lt;&gt;"/>
          </Variable>
          <Variable name="thresholdValue" type="string"/>
          <Variable name="timeout" type="int"/>
          <Variable name="timeUnit" type="enum">
            <Value desc="Seconds" value="seconds"/>
            <Value desc="Minutes" value="minutes"/>
            <Value desc="Hours" value="hours"/>
          </Variable>
        </FireRule>
        <ResetRule name="resetRule1">
          <Variable name="endpointId" type="string"/>
          <Variable name="clusterType" type="string"/>
          <Variable name="clusterId" type="string"/>
          <Variable name="attributeId" type="string"/>
          <Variable name="type" type="enum">
            <Value desc="Numeric" value="numeric"/>
            <Value desc="String" value="string"/>
          </Variable>
          <Variable name="operator" type="enum">
            <Value desc="&gt;" value="&gt;"/>
            <Value desc="&gt;=" value="&gt;="/>
            <Value desc="&lt;" value="&lt;"/>
            <Value desc="&lt;=" value="&lt;="/>
            <Value desc="=" value="="/>
            <Value desc="!=" value="&lt;&gt;"/>
          </Variable>
          <Variable name="thresholdValue" type="string"/>
          <Variable name="timeout" type="int"/>
          <Variable name="timeUnit" type="enum">
            <Value desc="Seconds" value="seconds"/>
            <Value desc="Minutes" value="minutes"/>
            <Value desc="Hours" value="hours"/>
          </Variable>
        </ResetRule>
      </Rules>
    </almtRules>
    <almtResourceList>DataPoint,AlarmStatus</almtResourceList>
  </AlarmTemplate>
```

Alarm template ID 8: Subscription usage

```
  <AlarmTemplate>
    <almtId>8</almtId>
    <almtName>Subscription Usage</almtName>
    <almtDescription>Fires when subscription usage exceeds a certain
threshold</almtDescription>
    <grpId>1</grpId>
    <almtTopic>Alarm.SubscriptionUsage</almtTopic>
    <almtScopeOptions>
      <ScopingOptions>
        <Scope name="Group"/>
        <Scope name="Global"/>
        <Scope name="Device"/>
```

```
      </ScopingOptions>
    </almtScopeOptions>
    <almtRules>
      <Rules>
        <FireRule name="fireRule1" uiView="SubscriptionFireRule">
          <Variable name="svcId" type="int"/>
          <Variable name="thresholdValue" type="numeric"/>
          <Variable name="metric" type="string"/>
          <Variable name="unit" type="string"/>
        </FireRule>
        <ResetRule name="resetRule1">
        </ResetRule>
      </Rules>
    </almtRules>
    <almtResourceList>SubscriptionUseCore,AlarmStatus</almtResourceList>
  </AlarmTemplate>
```

Alarm template ID 9: Data point condition

```
  <AlarmTemplate>
    <almtId>9</almtId>
    <almtName>DataPoint condition</almtName>
    <almtDescription>Fires when data point usage conditions given below is
met</almtDescription>
    <grpId>1</grpId>
    <almtTopic>Alarm.DataPointConditionMatch</almtTopic>
    <almtScopeOptions>
      <ScopingOptions>
        <Scope name="Resource"/>
      </ScopingOptions>
    </almtScopeOptions>
    <almtRules>
      <Rules>
        <FireRule name="fireRule1">
          <Variable name="type" type="enum">
            <Value desc="Numeric" value="numeric"/>
            <Value desc="String" value="string"/>
          </Variable>
          <Variable name="operator" type="enum">
            <Value desc="&gt;" value="&gt;"/>
            <Value desc="&gt;=" value="&gt;="/>
            <Value desc="&lt;" value="&lt;"/>
            <Value desc="&lt;=" value="&lt;="/>
            <Value desc="=" value="="/>
            <Value desc="!=" value="&lt;&gt;"/>
          </Variable>
          <Variable name="thresholdValue" type="string"/>
          <Variable name="timeout" type="int"/>
          <Variable name="timeUnit" type="enum">
            <Value desc="Seconds" value="seconds"/>
            <Value desc="Minutes" value="minutes"/>
            <Value desc="Hours" value="hours"/>
          </Variable>
        </FireRule>
        <ResetRule name="resetRule1">
          <Variable name="type" type="enum">
            <Value desc="Numeric" value="numeric"/>
            <Value desc="String" value="string"/>
```

```
          </Variable>
          <Variable name="operator" type="enum">
            <Value desc="&gt;" value="&gt;"/>
            <Value desc="&gt;=" value="&gt;="/>
            <Value desc="&lt;" value="&lt;"/>
            <Value desc="&lt;=" value="&lt;="/>
            <Value desc="=" value="="/>
            <Value desc="!=" value="&lt;&gt;"/>
          </Variable>
          <Variable name="thresholdValue" type="string"/>
          <Variable name="timeout" type="int"/>
          <Variable name="timeUnit" type="enum">
            <Value desc="Seconds" value="seconds"/>
            <Value desc="Minutes" value="minutes"/>
            <Value desc="Hours" value="hours"/>
          </Variable>
        </ResetRule>
      </Rules>
    </almtRules>
    <almtResourceList>DataPoint,AlarmStatus</almtResourceList>
  </AlarmTemplate>
```

Alarm template ID 10: Missing data point

```
  <AlarmTemplate>
    <almtId>10</almtId>
    <almtName>Missing DataPoint</almtName>
    <almtDescription>Fires when a data points are not reported within the
specified time</almtDescription>
    <grpId>1</grpId>
    <almtTopic>Alarm.MissingDataPoint</almtTopic>
    <almtScopeOptions>
      <ScopingOptions>
        <Scope name="Resource"/>
      </ScopingOptions>
    </almtScopeOptions>
    <almtRules>
      <Rules>
        <Description>Note: Alarm will not be registered until the first DataPoint
is sent after the Alarm is created or edited.</Description>
        <FireRule name="fireRule1">
          <Variable name="uploadInterval" type="int"/>
          <Variable name="uploadTimeUnit" type="enum">
            <Value desc="Minutes" value="minutes"/>
            <Value desc="Hours" value="hours"/>
          </Variable>
          <Variable name="readingInterval" type="int"/>
          <Variable name="readingTimeUnit" type="enum">
            <Value desc="Seconds" value="seconds"/>
            <Value desc="Minutes" value="minutes"/>
            <Value desc="Hours" value="hours"/>
          </Variable>
        </FireRule>
        <ResetRule name="resetRule1">
        </ResetRule>
      </Rules>
    </almtRules>
```

```
    <almtResourceList>DataPoint,AlarmStatus</almtResourceList>
  </AlarmTemplate>
```

Alarm template ID 11: Missing DIA channel data point

```
  <AlarmTemplate>
    <almtId>11</almtId>
    <almtName>Missing DiaChannel DataPoint</almtName>
    <almtDescription>Fires when devices have not reported DIA channel data within
the specified time</almtDescription>
    <grpId>1</grpId>
    <almtTopic>Alarm.MissingDiaChannelDataPoint</almtTopic>
    <almtScopeOptions>
      <ScopingOptions>
        <Scope name="Group"/>
        <Scope name="Device"/>
      </ScopingOptions>
    </almtScopeOptions>
    <almtRules>
      <Rules>
        <Description>Note: Alarm will not be registered until the first DataPoint
is sent after the Alarm is created or edited.</Description>
        <FireRule name="fireRule1">
          <Variable name="instanceName" type="string"/>
          <Variable name="channelName" type="string"/>
          <Variable name="uploadInterval" type="int"/>
          <Variable name="uploadTimeUnit" type="enum">
            <Value desc="Minutes" value="minutes"/>
            <Value desc="Hours" value="hours"/>
          </Variable>
          <Variable name="readingInterval" type="int"/>
          <Variable name="readingTimeUnit" type="enum">
            <Value desc="Seconds" value="seconds"/>
            <Value desc="Minutes" value="minutes"/>
            <Value desc="Hours" value="hours"/>
          </Variable>
        </FireRule>
        <ResetRule name="resetRule1">
        </ResetRule>
      </Rules>
    </almtRules>
    <almtResourceList>DataPoint,AlarmStatus</almtResourceList>
  </AlarmTemplate>
```

Alarm template ID 12: Missing smart energy data point

```
  <AlarmTemplate>
    <almtId>12</almtId>
    <almtName>Missing Smart Energy DataPoint</almtName>
    <almtDescription>Fires when devices have not reported Smart Energy data
within the specified time</almtDescription>
    <grpId>1</grpId>
    <almtTopic>Alarm.MissingSmartEnergyDataPoint</almtTopic>
    <almtScopeOptions>
      <ScopingOptions>
        <Scope name="Group"/>
        <Scope name="Device"/>
        <Scope name="XbeeNode"/>
```

```
      </ScopingOptions>
    </almtScopeOptions>
    <almtRules>
      <Rules>
        <Description>Note: Alarm will not be registered until the first DataPoint
is sent after the Alarm is created or edited.</Description>
        <FireRule name="fireRule1">
          <Variable name="endpointId" type="string"/>
          <Variable name="clusterType" type="string"/>
          <Variable name="clusterId" type="string"/>
          <Variable name="attributeId" type="string"/>
          <Variable name="uploadInterval" type="int"/>
          <Variable name="uploadTimeUnit" type="enum">
            <Value desc="Minutes" value="minutes"/>
            <Value desc="Hours" value="hours"/>
          </Variable>
          <Variable name="readingInterval" type="int"/>
          <Variable name="readingTimeUnit" type="enum">
            <Value desc="Seconds" value="seconds"/>
            <Value desc="Minutes" value="minutes"/>
            <Value desc="Hours" value="hours"/>
          </Variable>
        </FireRule>
        <ResetRule name="resetRule1">
        </ResetRule>
      </Rules>
    </almtRules>
    <almtResourceList>DataPoint,AlarmStatus</almtResourceList>
  </AlarmTemplate>
```

CarrierAuth

Use the CarrierAuth web service to get, configure, modify, or delete carrier account credentials.

URI

```
http://<hostname>/ws/CarrierAuth
```

Formats

HTTP method	Format	Description
GET	/ws/CarrierAuth	Get a list of all configured carrier accounts. **Note** Password information is not returned.
POST	/ws/CarrierAuth	Configure authorization information for a carrier account.
PUT	/ws/CarrierAuth/{caId}	Update carrier authorization information for an existing carrier account.
DELETE	/ws/CarrierAuth/{caId}	Delete carrier authorization information for a carrier account.

Elements

caId

Identifier associated with a specific carrier authentication. A unique identifier is returned for each CarrierAuth request.

cstId

Remote Manager identifier for the customer.

prvName

Cellular service provider name. Options include: ATT, DeutscheTelekom, Rogers, Telefonica, Verizon, or Vodafone.

caUserName

Username associated with the carrier account. This is the username provided by your business account carrier that you used to set up the carrier account within Remote Manager.

caPassword

Password for the cellular service account. This password was provided by your business account carrier.

caUpdateTime

Date and time in ISO 8601 format when your carrier account information was last updated.

caLicenseKey1

Admintrator license key required for AT&T, Rogers, and Telefonica.

Example: Get a list of carrier accounts

The following example shows how to get a list of configured carrier accounts for your Remote Manager account.

Note Password information is not retured.

Request

```
GET ws/CarrierAuth
```

Response

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
  <resultTotalRows>2</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>2</resultSize>
  <requestedSize>1000</requestedSize>
```

```
  <remainingSize>0</remainingSize>
  <CarrierAuth>
    <caId>2</caId>
    <cstId>3</cstId>
    <prvName>ATT</prvName>                                  <!-- carrier account
#1 -->
    <caUserName>exampleUser</caUserName>
    <caUpdateTime>2012-10-15T15:17:00.000Z</caUpdateTime>
  </CarrierAuth>
  <CarrierAuth>
    <caId>67</caId>
    <cstId>3</cstId>
    <prvName>Vodafone</prvName>                             <!-- carrier account
#2 -->
    <caUserName>exampleUser</caUserName>
    <caUpdateTime>2012-10-31T18:55:00.000Z</caUpdateTime>
  </CarrierAuth>
</result>
```

Example: Configure carrier account credentials

The following example shows how to configure credentials for an AT&T account.

```
POST /ws/CarrierAuth
```

```
<CarrierAuth>
   <prvName>ATT</prvName>
   <caUserName>ExampleUser</caUserName>
   <caPassword>123</caPassword>
   <caLicenseKey1>123</caLicenseKey1>
 </CarrierAuth>
```

Example: Update a carrier account

The following example shows how to insert a caId element in an existing AT&T account.

```
PUT /ws/CarrierAuth
```

```
<CarrierAuth>
   <caId>7</caId>
   <prvName>ATT</prvName>
   <caUserName>exampleUser</caUserName>
   <caPassword>123</caPassword>
   <caLicenseKey1>123</caLicenseKey1>
 </CarrierAuth>
```

Example: Delete a carrier account

The following example shows how to delete a carrier account.

```
DELETE ws/CarrierAuth/{subscription_id}
```

Replace subscription_id with the subscription ID of the account you want to delete.

DataPoint

Note The DataPoint API is a pre-version 1 API used to get, create, modify, or delete data points. Data points created by the API are supported. However, when creating new data streams, use the v1/streams API.

The DataPoint web service lists, creates, or deletes data points within a data stream.

URI

```
http://<hostname>/ws/DataPoint
```

Formats

Method	Format	Description
GET	/ws/DataPoint/{streamId}	List all data points for a data stream.
POST	/ws/DataPoint/{streamId}	Create one or more data points in a data stream.
DELETE	/ws/DataPoint/{streamId}	Delete an existing data point within a data stream.

Elements

id

Identifier for the data point.

cstId

Remote Manager identifier for the customer.

streamId

Full path for the stream that contains the data points. Typically this is the data stream that the data point belongs to, but if you are using replication (forwardTo) it may be different.

timestamp

Client-assigned timestamp. If there is no client-assigned timestamp, the serverTimestamp value is used.

serverTimestamp

Server-assigned timestamp that indicates when the data point was stored on the server. Not writable by the client.

data

Data value for the data point.

description

Description of the data.

quality

User-defined 32-bit integer value representing the quality of the data in the data point.

location

Geo-location information associated with the data point which indicates the location when the data point was recorded. Geo-location is represented as a comma-delimited list of floats in order of lat, long, elevation (degrees, degrees, meters).

dataType

Type of data stored in the data stream.

- Integer: data can be represented with a network (= big-endian) 32-bit two's-complement integer
- Long: data can be represented with a network (= big-endian) 64-bit two's complement integer
- Float: data can be represented with a network (= big-endian) 32-bit IEEE754 floating point
- Double: data can be represented with a network (= big-endian) 64-bit IEEE754 floating point
- String: UTF-8
- Binary
- Unknown

units

User-defined name for the units in which data is reported.

forwardTo

Comma-delimited list of data streams to which to forward the data points.

Parameters

Name	Type	Description
startTime	timestamp	Start time (inclusive) in ISO 8601 or epoch (long).
endTime	timestamp	End time (exclusive) in ISO 8601 or epoch (long).
timeline	string	Timestamps to use in the request: client or server. The default is client.
pageCursor	string	Cursor to get the next page of devices. Omit on initial call.
size	integer	Number of items to return. The maximum and default is 1000.
order	string	Return streams ordered by ID (asc | desc). The default is ascending (asc).
timezone	string	Timezone in which to calculate rollups. Applies only to rollups with intervals of day or longer.
rollupInterval	string	Rollup interval: half, hour, day, week, or month. The default is hour.
rollupMethod	string	Rollup method: sum, average, min, max, count, or standarddev. The default is average.

Direct device uploads

Devices can upload directly to data streams over any of the existing transports (TCP, UDP, SMS, and Satellite). The path specified in the data service message begins with **DataPoint** and the rest of the message is mapped to a data stream appended to the device ID.

For example, if the device sends a data point file specifying the filename **DataPoint/temp1**, the data point is added to the data stream **<device-id>/temp1**. The file must follow one of the expected formats and must specify the format via the file extension. The following types are supported for a given extension:

Format	Extension	Description
XML	.xml	XML representation same as the /ws/DataPoint interface.
CSV	.csv	Comma separated list. One data point per line with details separated by commas.
Binary	.bin	Whatever the content of the uploaded data is directly inserted to a single data point.

Data limits related to direct device uploads

To maximize the speed and throughput of Remote Manager, limitations have been imposed on device uploads.

- Maximum number of data points allowed per request: 250
- Maximum size of Send Data requests: 2MB
- Maximum size of replies to Device Requests: 2MB
- Maximum number of binary data points allowed: 64KB

Note The Description field for a data point does not display in the Remote Manager UI Data Streams view.

When devices push data points up to Remote Manager, the description included refers to the data point, not the data stream. To view the description, you must retrieve data point via web services.

XML

XML format uses the same format used in /ws/DataPoint PUT. The stream id is ignored since it is provided by the path. Also, any streams listed in the **forwardTo** field will be normalized to the device's stream. This is done to prevent one device from uploading data into another device's stream.

```
<DataPoint>
    <data>42</data>
    <!-- Everything below this is optional -->
    <description>Temperature at device 1</description>
    <location>0.0, 0.0, 0.0</location>
    <quality>99</quality>
    <dataType>float</dataType>
    <units>Kelvin</units>
</DataPoint>
```

For multiple data points in one message:

```
<list>
  <DataPoint>
    <data>42</data>
    <timestamp>1234566</timestamp>
  </DataPoint>
  <DataPoint>
    <data>43</data>
  </DataPoint>
</list>
```

CSV

An optional upload format is to specify the data in UTF-8 encoded comma separated values. Each line (`'\n'` terminated) specifies a data point. The default order is:

```
DATA, TIMESTAMP, QUALITY, DESCRIPTION, LOCATION, DATATYPE, UNITS, FORWARDTO
```

Meaning the following file:

```
data, 1,99,"my description",,INTEGER,kelvins,"stream1,stream2"
data2,2,50,"my description"
data3,3,25,"my description"
```

Would create 3 data points, set the stream's units/type to kelvins/Integers, and have the data points with the data "data", "data2", and "data3", using the epoch timestamps of 1, 2, and 3.

Note that location was omitted in the above example. You can omit values by leaving them empty or stopping before the end. For example:

Empty values:data,1,,,99
Ending early:data,1

Order can be overridden. You can define a header on the first line by starting it with a '#' character, for example:

```
#TIMESTAMP,DATA
1, data
2, data2
3, data3
```

Will create 3 data points 1ms apart starting at epoch (1970).

Multiple datapoints for multiple streams from a device can be inserted in one message using the STREAMID value. When the STREAMID value is specified, the file name is no longer used for the stream name.

For example:

```
#STREAMID,DATA,TIMESTAMP
sensor1/port1,97,1
sensor1/port2,98,1
sensor2/port1,42,1
sensor2/port2,0,2
```

Will create 4 data points, one in each of 4 separate streams for the device. The first 3 data points are at 1ms after the epoch (1970) and the final data point is 1ms later.

The XML version is as follows:

```
<list>
<DataPoint><streamId>sensor1/port1</streamId><data>97</data><timestamp>1</timesta
mp></DataPoint>
<DataPoint><streamId>sensor1/port2</streamId><data>98</data><timestamp>1</timesta
mp></DataPoint>
<DataPoint><streamId>sensor2/port1</streamId><data>42</data><timestamp>1</timesta
mp></DataPoint>
<DataPoint><streamId>sensor2/port2</streamId><data>0</data><timestamp>2</timestam
p></DataPoint>
</list>
```

Binary Concise Alternative Format

The disadvantage to using the XML format is that it is very verbose. This binary alternative format can be used to be more concise. You can specify a simple value instead of XML or CSV data. When the value is pushed to /DataPoint, it is stored in complete as-is in time-series data (in the exact binary format as provided). For details on endianness, bit lengths, and so on for supported data types see the dataType in the Data Streams section. However, data types are not required. Data can be 1 byte status indicators or 10k images but Remote Manager will not be able to provide rollups on things which do not use the specified formats.

For instance, the following data service message:

path: /DataPoint/temp1.bin

content: 42

Will result in a new data point with a value of "42" (in binary).

Note: The binary concise mechanism has the following limitations:

- Only single values can be uploaded per data service message
- Data must be smaller than 64k

Deciding which format to use when inserting data

Whitespace characters for the data value are preserved in all formats. Use quotes around the string for CSV format to preserve break lines. For binary data, we recommend you to use binary concise format. Binary concise format however can't be used to create multiple data points in a single request. To create multiple binary data points in a single request, we recommend you to use a base64 encoded string.

DataStream

Note The DataStreams API is a pre-version 1 API used the get, create, modify, or delete data streams. Data streams created by the API are supported; when creating a new data stream, use the v1/streams API.

The DataStream web services creates, modifies, or deletes a data stream.

URI

```
http://<hostname>/ws/DataStream
```

Formats

Method	Format	Description
GET	/ws/DataStream	List all data streams.
POST	/ws/DataStream	Create one or more data streams.
PUT	/ws/DataStream	Create or update a data stream.
DELETE	/ws/DataStream/{streamId}	Delete a data stream.

Elements

cstId

Remote Manager identifier for the customer.

streamId

Full path for the stream that contains the data points. Typically this is the data stream that the data point belongs to, but if you are using replication (forwardTo) it may be different.

dataType

Type of data stored in the data stream.

- Integer: data can be represented with a network (= big-endian) 32-bit two's-complement integer

- Long: data can be represented with a network (= big-endian) 64-bit two's complement integer
- Float: data can be represented with a network (= big-endian) 32-bit IEEE754 floating point
- Double: data can be represented with a network (= big-endian) 64-bit IEEE754 floating point
- String: UTF-8
- Binary
- Unknown

units

User-defined name for the units in which data is reported.

description

Description of the data.

forwardTo

Comma-delimited list of data streams to which to forward the data points.

dataTtl

Time to live (TTL) in seconds for data points stored in the data stream. A data point expires after the configured amount of time and is automatically deleted.

rollupTtl

Time to live (TTL) in seconds for the aggregate roll-ups of data points stored in the stream. A roll-up expires after the configured amount of time and is automatically deleted.

currentValue

Information about the last recorded data point (not writeable in PUT or POST requests).

Field	Description
id	Identifier for the data point.
timestamp	Data point client timestamp.
serverTimestamp	Timestamp when data point was received by the server.
data	Data value of the data point.
description	Data point description.
quality	User-defined 32-bit integer value representing the quality of the data in the data point.
location	Geo-location information associated with the data point which indicates the location when the data point was recorded. Geo-location is represented as a comma-delimited list of floats in order of lat, long, elevation (degrees, degrees, meters).

Parameters

Name	Type	Description
pageCursor	string	Page cursor returned from a previous request that can be used to retrieve the next page of data. Omit on initial call.
size	integer	Maximum number of items to return. The maximum and default is 1000.
category	string	Return streams for the specified category: data, metrics, management, or carrier. If you do not use the category parameter, streams for all categories are returned.

DeviceCore

Use the DeviceCore web service to create, register, modify, or delete Remote Manager devices or to retrieve information about a registered device. You can retrieve settings, connection information, and state information for a registered device.

URI

```
http://<hostname>/ws/DeviceCore
```

Formats

HTTP method	Format	Description
GET	/ws/DeviceCore	Get a list of devices provisioned in your account.
POST	/ws/DeviceCore/{devConnectwareId}	Add or register a device in your account.
PUT	/ws/DeviceCore/[{id}|{devConnectwareId}]	Add descriptive text fields for the device.
DELETE	/ws/DeviceCore/[{id}	Delete a device from your account.

Elements

cstId

Remote Manager identifier for the customer.

devCellularModemId

Modem identifier of the device.

devConnectwareId

Device identifier of the device.

devEffectiveStartDate

Date the device was provisioned in Remote Manager.

devInstallCode

Installation code for the device. An installation code is required for any device manufactured with an associated installation code.

- If you attempt to add a device that requires an installation code with a missing or incorrect code, you receive an HTTP status 400 error code along with a message describing the error.
- If you are adding multiple devices and one or more of the device installation code is missing or incorrect, you receive an HTTP status 207 error along with a message describing the error.

devMac

MAC address for the device.

devRecordStartDate

Date the device record was created.

devTerminated

False if the device is currently in the customer account.

dpConnectionStatus

Connection status for the device

Value	Description
0	Disconnected
1	Connected

dpContact

Contact setting from the device.

dpCurrentConnectPw

Password for the device to connect to Remote Manager. If set, the device must provide the password to connect.

dpDescription

Description setting from the device.

dpDeviceType

Manufacturer-assigned device type, such as ConnectPort X2.

dpFirmwareLevel

Integer that represents the firmware level. For example, 34209795.

dpFirmwareLevelDesc

String value that represents the firmware level. For example, 2.10.0.3.

dpGlobalIp

IP address from which the device connected in IPv4 format.

dpLastConnectTime

Date the device last connected to Remote Manager. For example, 2010-07-21T15:20:00Z.

dpLastKnownIp

IP address last reported by the device in IPv4 format.

dpLocation

Location setting from the device.

dpMapLat

Map latitude setting from the device.

dpMapLong

Map longitude setting from the device.

dpPanId

PanId setting from the device.

dpRestrictedStatus

Indicates restrictions on the device for connecting to Remote Manager:

Value	Description
0	Unrestricted
2	Restricted
3	Untrusted

dpServerId

Identifier of the server to which the device is currently connected.

dpTags

Comma-delimited set of user-defined tags.

dpUserMetaData

User-specified free-form text field.

dvVendorId

Integer that identifies the manufacturing vendor.

grpId

Remote Manager identifier for the customer group.

grpPath

Full path name of the specified group. For PUT or POST requests, if the specified group does not exist, Remote Manager creates the group.

id

Unique identifier for the device that consists of the following elements:

- devId: System-generated identifier for the device.
- devVersion: Version for the device. A value of 0 indicates the most current version.

provisionId

Randomly-generated identifier used to provision the device. This identifier must be used in place of devConnectwareId and you must supply a vendor ID.

xpExtAddr

ZigBee 64-bit extended address from the device.

DeviceInterface

Use the DeviceInterface web service to get a list of devices and associated networks.

URI

```
http://<hostname>/ws/DeviceInterface
```

Formats

HTTP method	Format	Description
GET	/ws/DeviceInterface	Get a list of devices and associated networks.

Elements

None

Example: Get a list of devices and associated networks

The following example shows how to get a list of devices and associated networks.

Request

```
GET /ws/DeviceInterface
```

Response (abbreviated)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
   <resultTotalRows>3585</resultTotalRows>
   <requestedStartRow>0</requestedStartRow>
   <resultSize>1000</resultSize>
   <requestedSize>1000</requestedSize>
   <remainingSize>2585</remainingSize>
   <DeviceInterface>
      <id>
         <devId>664928</devId>
         <devVersion>0</devVersion>
         <niId>0</niId>
         <niVersion>0</niVersion>
      </id>
      <devRecordStartDate>2014-09-15T16:27:00.000Z</devRecordStartDate>
      <devConnectwareId>00000000-00000000-000000FF-FF000088</devConnectwareId>
      <cstId>2</cstId>
      <grpId>2</grpId>
      <devEffectiveStartDate>2014-09-15T16:27:00.000Z</devEffectiveStartDate>
      <devTerminated>false</devTerminated>
   </DeviceInterface>
   <DeviceInterface>
      <id>
         <devId>1205698</devId>
         <devVersion>0</devVersion>
         <niId>0</niId>
         <niVersion>0</niVersion>
      </id>
      <devRecordStartDate>2014-09-07T14:19:00.000Z</devRecordStartDate>
      <devMac>00:40:9A:DA:01:E5</devMac>
      <devConnectwareId>00000000-00000000-00409AFF-FFDA01E5</devConnectwareId>
      <cstId>2</cstId>
      <grpId>1326</grpId>
      <devEffectiveStartDate>2014-05-08T18:44:00.000Z</devEffectiveStartDate>
      <devTerminated>false</devTerminated>
   </DeviceInterface>
   <DeviceInterface>
      <id>
         <devId>1205699</devId>
         <devVersion>0</devVersion>
         <niId>0</niId>
         <niVersion>0</niVersion>
      </id>
      <devRecordStartDate>2014-09-12T15:49:00.000Z</devRecordStartDate>
      <devMac>00:40:9A:DA:01:E7</devMac>
      <devConnectwareId>00000000-00000000-00409AFF-FFDA01E7</devConnectwareId>
```

```
      <cstId>2</cstId>
      <grpId>2</grpId>
      <devEffectiveStartDate>2014-05-08T18:44:00.000Z</devEffectiveStartDate>
      <devTerminated>false</devTerminated>
   </DeviceInterface>
   <DeviceInterface>
      <id>
         <devId>1205700</devId>
         <devVersion>0</devVersion>
         <niId>0</niId>
         <niVersion>0</niVersion>
      </id>
      <devRecordStartDate>2014-09-07T14:19:00.000Z</devRecordStartDate>
      <devMac>00:40:9A:DA:01:E6</devMac>
      <devConnectwareId>00000000-00000000-00409AFF-FFDA01E6</devConnectwareId>
      <cstId>2</cstId>
      <grpId>1326</grpId>
      <devEffectiveStartDate>2014-05-08T18:44:00.000Z</devEffectiveStartDate>
      <devTerminated>false</devTerminated>
   </DeviceInterface>
   <DeviceInterface>
      <id>
         <devId>1205701</devId>
         <devVersion>0</devVersion>
         <niId>0</niId>
         <niVersion>0</niVersion>
      </id>
      <devRecordStartDate>2014-09-07T14:19:00.000Z</devRecordStartDate>
      <devMac>00:40:9A:DA:01:E8</devMac>
      <devConnectwareId>00000000-00000000-00409AFF-FFDA01E8</devConnectwareId>
      <cstId>2</cstId>
      <grpId>1326</grpId>
      <devEffectiveStartDate>2014-05-08T18:44:00.000Z</devEffectiveStartDate>
      <devTerminated>false</devTerminated>
   </DeviceInterface>
```

DeviceMetaData

Use the DeviceMetaData to manage embedded device view descriptors not directly available from a device.

URI

```
http://<hostname>/ws/DeviceMetaData
```

Formats

HTTP Method	Format	Description
GET	/ws/DeviceMetaData	Display a list of view descriptors for a vendor ID.
POST	/ws/DeviceMetaData	Add a view descriptor.
PUT	/ws/DeviceMetaData	Update a view descriptor.
DELETE	/ws/DeviceMetaData	Delete a view descriptor.

Elements

dmId

Unique identifier for the metadata.

dvVendorId

Integer that identifies the manufacturing vendor.

dmDeviceType

Name of the device type.

dmProductId

Identifier of the product to which this metadata corresponds.

dmFirmwareId

Identifier of the firmware to which this metadata corresponds.

dmVersion

Firmware version to which the metadata corresponds.

dmName

Defines the descriptor type. Must be descriptor/ui.

dmCompressed

Indicates whether the metadata is compressed. Typically, metadata is not compressed.

dmData

Metadata contents.

DeviceVendor

Use the DeviceVendor web service to get a list of vendor identifiers available for your account, update the group into which new devices are provisioned, or update the default restriction status for new devices.

To see your vendor ID or register for an ID:

1. Log in to your Remote Manager account.
2. Click **Admin > Account Settings > My Account**.
   - If you have already registered a vendor ID, the vendor ID is displayed, as well as the provisioning configuration.
   - If you have not registered for a vendor ID, click **Register for new vendor id** and a vendor ID is assigned to your account. Refresh the account page to see the assigned vendor ID.

URI

```
http://<hostname>/ws/DeviceVendor
```

Formats

HTTP Method	Format	Description
GET	/ws/DeviceVendor	Retrieve vendor IDs available for your account.
POST	/ws/DeviceVendor	Register a vendor ID to use for device development.
PUT	/ws/DeviceVendor	Update grpPath or dpRestrictedStatus elements for a vendor.

Elements

dvVendorId

Integer that identifies the manufacturing vendor.

dvVendorIdDesc

Hexadecimal representation of the Vendor ID.

cstId

Remote Manager identifier for the customer.

dvDescription

Text description for the vendor ID.

dvRegistrationDate

Date when the Vendor ID was registered.

grpPath

Name of a group into which new auto-provisioned devices are put by default. `<grpPath disabled="true"/>` disables auto-provisioning. If you create a new device ID by performing a POST to ws/DeviceVendor, you can specify a grpPath that overrides the default group path.

dpRestrictedStatus

Indicates restrictions on the device for connecting to Remote Manager:

Value	Description
0	Unrestricted
2	Restricted
3	Untrusted

DeviceVendorSummary

Use the DeviceVendorSummary web service to get a summary of device types for your vendor ID.

URI

```
http://<hostname>ws/DeviceVendorSummary
```

Formats

HTTP Method	Format	Description
GET	/ws/DeviceVendorSummary	Retrieve a list of device types associated with your vendor IDs.

Elements

dvVendorId

Integer that identifies the manufacturing vendor.

dmDeviceType

Name of the device type.

dvVendorIdDesc

Hexadecimal representation of the Vendor ID.

cstId

Remote Manager identifier for the customer.

dvDescription

Text description for the vendor ID.

dmUiDescriptorCount

Indicates the number of UI descriptors for the device type.

FileData

Use the FileData web service to query or locate one or more files based on file metadata, such as the name, type, storage path, size, or modification date.

URI

```
http://<hostname>/ws/FileData
```

Formats

HTTP method	Format	Description
GET	/ws/FileData	Get a paged list of file metadata for all of your files.
PUT	/ws/FileData/<fdPath>/<fdName>	Upload or change a file or folder in your account.
DELETE	/ws/FileData/<fdPath>/<fdName>	Delete a file or folder from your account.

Elements

fdPath

Specifies the hierarchical path to the file. Use the tilde character (**~**) to indicate your home directory.

fdName

Specifies the name of the file.

cstId

Remote Manager identifier for the customer.

fdCreatedDate

Specifies the date the file was first uploaded to Remote Manager (ISO 8601 standard format).

fdLastModifiedDate

Specifies the date the file was last modified (ISO 8601 standard format).

fdContentType

Specifies the type of data stored in the file.

fdSize

Specifies the size of the file in bytes.

fdType

Specifies the file type: file or directory.

The default is file.

[fdData]

Includes the Base64-encoded content of the file. A tool to encode and decode Base64 data is available here: http://ostermiller.org/calc/encode.html.

Example: Get all file metadata

The following example shows how to get a paged list of file metadata for all of your files.

Request

```
GET /ws/FileData
```

Response (abbreviated)

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
    <resultTotalRows>455747</resultTotalRows>
    <requestedStartRow>0</requestedStartRow>
    <resultSize>1000</resultSize>
    <requestedSize>1000</requestedSize>
    <remainingSize>454747</remainingSize>
    <FileData>
        <id>
            <fdPath>/db/SB723050334974_Digi_International/00000000-00000000-
00409DFF-FF640005/</fdPath>
            <fdName>RPC_response-1297463631.0-0001-received_attribute_
report.xml</fdName>
        </id>
        <cstId>3439</cstId>
        <fdCreatedDate>2011-02-11T22:34:25Z</fdCreatedDate>
        <fdLastModifiedDate>2011-02-11T22:34:25Z</fdLastModifiedDate>
        <fdContentType>application/xml</fdContentType>
        <fdSize>506</fdSize>
        <fdType>file</fdType>
    </FileData>...<FileData>
        <id>
            <fdPath>/db/SB723050334974_Digi_International/00000000-00000000-
00409DFF-FF640005/</fdPath>
            <fdName>RPC_response-1297463631.0-0003-received_attribute_
report.xml</fdName>
        </id>
        <cstId>3439</cstId>
        <fdCreatedDate>2011-02-11T22:34:25Z</fdCreatedDate>
        <fdLastModifiedDate>2011-02-11T22:34:25Z</fdLastModifiedDate>
        <fdContentType>application/xml</fdContentType>
        <fdSize>506</fdSize>
        <fdType>file</fdType>
    </FileData>
</result>
```

Example: Get files based on conditions

The following examples show how to get files based on conditions.

Example 1: Get files written after a specified date:

```
GET /ws/FileData?condition=fdType='file' and fdLastModifiedDate>'2013-12-
06T14:50:00.000Z'
```

Example 2: Get files that match name patterns using wildcards

The following example returns all files whose name starts with 'sample' and ends with 'gas' that were written to Remote Manager after the specified date.

```
GET /ws/FileData?condition=fdName like 'sample%25gas' and fdType='file' and
fdLastModifiedDate>'2013-12-06T14:50:00.000Z'
```

Example: Get files and embed contents in the result

The following example shows how to use the embed="true" option to embed the content of the file in the results in Base64 format.

Request

```
GET /ws/FileData?condition=fdPath='~/00000000-00000000-00409DFF-FF640005/' and
fdType='file'
and fdLastModifiedDate>'2010-11-24T22:25:04Z'&embed=true
```

Response

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
    <resultTotalRows>1264</resultTotalRows>
    <requestedStartRow>0</requestedStartRow>
    <resultSize>1000</resultSize>
    <requestedSize>1000</requestedSize>
    <remainingSize>264</remainingSize>
    <FileData>
        <id>
            <fdPath>/db/SB723050334974_Digi_International/00000000-00000000-
00409DFF-FF640005/</fdPath>
            <fdName>RPC_response-1297463631.0-0001-received_attribute_
report.xml</fdName>
        </id>
        <cstId>3439</cstId>
        <fdCreatedDate>2011-02-11T22:34:25Z</fdCreatedDate>
        <fdLastModifiedDate>2011-02-11T22:34:25Z</fdLastModifiedDate>
        <fdContentType>application/xml</fdContentType>
        <fdSize>506</fdSize>
        <fdType>file</fdType>
        <fdData>....</fdData>
    </FileData>...<FileData>
        <id>
            <fdPath>/db/SB723050334974_Digi_International/00000000-00000000-
00409DFF-FF640005/</fdPath>
            <fdName>attribute_report.xml</fdName>
        </id>
        <cstId>3439</cstId>
        <fdCreatedDate>2011-02-11T22:34:25Z</fdCreatedDate>
        <fdLastModifiedDate>2011-02-11T22:34:25Z</fdLastModifiedDate>
        <fdContentType>application/xml</fdContentType>
        <fdSize>506</fdSize>
        <fdType>file</fdType>
        <fdData>....</fdData>
    </FileData>
</result>
```

FileDataCore

Use the FileDataCore web service to get a count and listing of all files stored on Remote Manager for your account. FileDataCore does not return file contents. To retrieve file contents, use the FileData web service.

URI

```
http://<hostname>/ws/FileDataCore
```

Format

HTTP method	Format	Description
GET	/ws/FileDataCore	Get a summary count and listing of all files for your Remote Manager account.

Elements

None

FileDataHistory

Use the FileDataHistory web service to display activity history for files you have uploaded to a device. You can display archive history only for files that have a flag set to archive to the history table when the file was originally uploaded.

URI

```
http://<hostname>/ws/FileDataHistory
```

Formats

HTTP method	Format	Description
GET	/ws/FileDataHistory	Display activity history for each file you have uploaded to a device.

Elements

None

Group

Use the Group web service to retrieve information about groups in your Remote Manager account.

URI

```
http://<hostname>/ws/Group
```

Formats

HTTP method	Format	Description
GET	/ws/Group	Get a list of all groups in your Remote Manager account.

Elements

grpId

Remote Manager identifier for the customer group.

grpName

Name of the group.

grpDescription

Description of the group.

grpPath

Full path name of the specified group.

grpParentId

Integer representation of the group parent.

Monitor

Use the Monitor web service to monitor Remote Manager activity and push notifications to a client application. Each configured monitor specifies the Remote Manager events or activities to monitor, the notification mechanism, and the transport type (TCP, HTTP, or Polling).

Monitored events can include:

- **Data:** Data pushed into Remote Manager from remote gateways in the form of DataPoints (directly or through DIA or Smart Energy), FileData, and so on.
- **Status:** General status updates such as connection status, remote device availability, and so on.
- **Alarms:** Alarm status updates, such as when alarms are triggered or reset.

Note FileData and FileDataCore events are not published when the file size is larger than 120K. Delete operations for FileData events are never published.

The Monitor web service is available only for Remote Manager accounts with a subscription to the Push Monitor service.

For information on retrieving saved pushed notifications, see v1/monitors/history.

URI

```
http://<hostname>/ws/Monitor
```

Formats

HTTP method	Format	Description
GET	/ws/Monitor	Get a list of all configured monitors.
GET	/ws/Monitor/{monId}	Get details for a specific monitor.
POST	/ws/Monitor	Create a new monitor to push event notifications.

HTTP method	Format	Description
PUT	/ws/Monitor/{monId}	Update an existing monitor. Note that any PUT request to a monitor resets the monitor state.
DELETE	/ws/Monitor/{monId}	Delete a monitor.

Elements

monId

System-generated identifier for the monitor.

cstId

Remote Manager identifier for the customer.

monFormatType

Format for delivered event data:

- xml
- json

monTopic

One or more topics to monitor. Supported monitor topics include:

- Alarm
- AlarmStatus
- DataPoint
- DataStream
- DeviceCore
- FileData
- FileDataCore
- Job
- JobResult
- XbeeCore

Note The following monitor topics have been deprecated and should not be used: DiaChannelDataFull, XbeeAttributeDataCore, XbeeEventDataCore.

Note FileData and FileDataCore events are not published when the file size is larger than 120K. Delete operations for FileData events are not published.

Note DataStream updates publish changes to DataStream attributes only, not currentValues. To get changes for currentValue, monitor the DataPoint topic to get changes to the current value as they arrive.

To monitor	Specify
general topic	Resource name only. For example: `DataPoint` Monitors all DataPoint events.
specific resource	Resource name followed by the resource ID using standard REST slash conventions. For example: `DataPoint/00000000-00000000-00000000-00000000` Monitors DataPoint events reported by the specific device.
multiple topics	Comma-delimited list of topics. For example: `DataPoint,DeviceCore` Monitors all DataPoint and Device data for the current customer.

scope by operation	By default, all operations for the specified monitor topic are monitored. To limit the monitor topic to specific operations, prefix the monitor topic with the operation qualifier. Valid operations:
	■ C for create
	■ U for any update
	■ D for delete
	For example, to monitor update operations only for DeviceCore:
	`[operation=U]DeviceCore`
	To monitor create and update operations for DeviceCore:
	`[operation=C,U]DeviceCore`

scope by group	By default, all groups for the specified monitor topic are monitored. To limit the monitor topic to one or more groups, prefix the monitor topic with the group qualifier. For example: `[group=America,Columbia]DeviceCore`

To monitor	Specify
scope by operation and group	To use both the operation and the group qualifiers, prefix the monitor topic with both qualifiers: `[operation=C,U,D][group=America,Columbia]DeviceCore` **Note** You can prefix the qualifiers in any order.
special characters	URL encode the following special characters when specifying additional subtopic components: / (forward slash) % (percent sign) . (period) * (asterisk) [(open bracket)] (close bracket) When monitor topics are reported, components are URL encoded. This allows for easy parsing of monitor topics. The general procedure is to split the published topic string on the backslash (/) separator character and then URL decode the identified components.

monTransportType

Transport method used to deliver push notifications to the client application:

- **tcp:** Push notifications are sent using TCP. See TCP transport protocol.
- **http:** Push notifications are sent using HTTP. See HTTP/HTTPS transport protocol.
- **polling:** Push notifications are saved but not sent. See v1/monitors/history for information on retrieving polling monitor notifications.

monTransportUrl

For HTTP transport type only. URL of the customer web server. For http URLs, the default listening port is 80; for https URLs, the default listening port is 443.

monTransportToken

For HTTP transport type only. Credentials for basic authentication in the following format:

```
username:password
```

monTransportMethod

For HTTP transport type only. HTTP method to use for sending data: PUT or POST. The default is PUT.

monConnectTimeout

For HTTP transport type only. Time in milliseconds Remote Manager waits when attempting to connect to the destination http server. A value of 0 means use the system default of 5000 (5 seconds). Most monitors do not need to configure this setting.

monResponseTimeout

For HTTP transport type only. Time in milliseconds Remote Manager waits for a response for pushed events from the http server. A value of 0 means use the system default of 5000 (5 seconds). Most monitors do not need to configure this setting.

monAckOption

For TCP transport type only. Indicates whether the client will explicitly acknowledge TCP push events or allow Remote Manager to automatically acknowledge events when sent. Options include: explicit or off. The default is off.

monBatchSize

Specifies an upper bound on how many messages are aggregated before sending a batch. The default is 100.

monBatchDuration

Specifies an upper bound on the number of seconds messages are aggregated before sending. The default is 10.

monCompression

Keyword that specifies the method used to compress messages. Options include: zlib or none. The default is none. For zlib, the deflate algorithm is used to compress the data; use inflate to decompress the data.

Note For backwards compatibility, gzip is accepted as a valid keyword. Compression has always been done using the deflate algorithm.

monAutoReplayOnConnect

Boolean value that specifies whether Remote Manager replays any missed published events before any new published events are forwarded. True indicates missed published events are replayed. False indicates missed published events are not replayed. The default is false.

monDescription

Optional text field used to label or describe the monitor.

monLastConnect

Returned in the GET response. Specifies last connection time to the client application.

monLastSent

Returned in the GET response. Specifies the last message pushed to the client application.

monStatus

Returned in the GET response. Specifies the current connection status to the client application.

Status	Description
CONNECTING	For HTTP monitors only. Remote Manager is attempting to connect to the configured HTTP server. Once connected, the state changes to ACTIVE.
ACTIVE	Monitor is connected and publishing events.
INACTIVE	Monitor is not connected and events are not published or recorded.
SUSPENDED	For monitors with monAutoReplayOnConnect = True. Monitor has disconnected, but publish events are recorded for later replay.
DISABLED	For HTTP monitors only. If a monitor has not connected for 24 hours, the state is set to DISABLED, and publish events are not recorded for replay. A disabled monitor must be reconfigured via the Monitor web service.
DISCONNECT	Monitor is currently disconnecting, and events are not being published. For monitors with monAutoReplayOnConnect = True, events are recorded for later replay. (Dashboard shows status as Disconnecting.)

Any PUT request to a monitor resets the monitor state.

Example: List all monitors

The following example shows how to list all configured monitors.

Request

```
GET /ws/Monitor
```

Response

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
  <resultTotalRows>4</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>4</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <Monitor>
    <monId>148214</monId>
    <cstId>2</cstId>
    <monLastConnect>2014-07-09T22:01:41.187Z</monLastConnect>
    <monLastSent>2014-07-09T22:02:09.000Z</monLastSent>
    <monTopic>DeviceCore</monTopic>
    <monTransportType>tcp</monTransportType>
    <monFormatType>json</monFormatType>
    <monBatchSize>1</monBatchSize>
    <monCompression>zlib</monCompression>
    <monStatus>INACTIVE</monStatus>
    <monBatchDuration>60</monBatchDuration>
    <monLastSentUuid>ac59ee13-07b4-11e4-a573-fa163ef93b22</monLastSentUuid>
  </Monitor>
  <Monitor>
    <monId>148215</monId>
    <cstId>2</cstId>
    <monLastConnect>2014-07-21T21:24:02.507Z</monLastConnect>
    <monLastSent>2014-07-14T17:17:15.000Z</monLastSent>
    <monTopic>DeviceCore,XbeeCore</monTopic>
    <monTransportType>http</monTransportType>
    <monTransportUrl>https://google.com</monTransportUrl>
    <monFormatType>json</monFormatType>
    <monBatchSize>100</monBatchSize>
    <monCompression>none</monCompression>
    <monStatus>DISABLED</monStatus>
    <monBatchDuration>10</monBatchDuration>
    <monTransportMethod>PUT</monTransportMethod>
  </Monitor>
  <Monitor>
    <monId>148218</monId>
    <cstId>2</cstId>
    <monLastConnect>2014-07-21T20:41:52.350Z</monLastConnect>
    <monLastSent>2014-07-21T19:15:37.000Z</monLastSent>
    <monTopic>DeviceCore,AlarmStatus</monTopic>
    <monTransportType>tcp</monTransportType>
    <monFormatType>json</monFormatType>
    <monBatchSize>1</monBatchSize>
    <monCompression>none</monCompression>
```

```
    <monStatus>DISABLED</monStatus>
    <monBatchDuration>1</monBatchDuration>
    <monAutoReplayOnConnect>true</monAutoReplayOnConnect>
    <monLastSentUuid>6590870c-110b-11e4-b325-fa163ef93b22</monLastSentUuid>
</Monitor>
<Monitor>
  <monId>148447</monId>
  <cstId>2</cstId>
  <monLastConnect>2014-09-19T14:13:19.077Z</monLastConnect>
  <monLastSent>2014-09-17T18:24:08.317Z</monLastSent>
  <monTopic>JobResult</monTopic>
  <monTransportType>http</monTransportType>

<monTransportUrl>http://10.235.3.133:8080/profilemanager/monitor/push</monTranspo
rtUrl>
  <monFormatType>xml</monFormatType>
  <monBatchSize>1000</monBatchSize>
  <monCompression>none</monCompression>
  <monStatus>DISABLED</monStatus>
  <monBatchDuration>10</monBatchDuration>
  <monAutoReplayOnConnect>true</monAutoReplayOnConnect>
  <monTransportMethod>PUT</monTransportMethod>
  <monLastSentUuid>4ca204aa-3e71-11e4-8f05-fa163e6d4ac5</monLastSentUuid>
</Monitor>
</result>
```

Example: Create an HTTP monitor

The following sample shows how to create a simple HTTP monitor.

```
POST /ws/Monitor
```

```
<Monitor>
  <monTopic>DeviceCore,XbeeCore</monTopic>
  <monTransportType>http</monTransportType>
  <monTransportUrl>https://your web site url</monTransportUrl>
  <monTransportToken>username:password</monTransportToken>
  <monTransportMethod>PUT</monTransportMethod>
  <monFormatType>json</monFormatType>
  <monBatchSize>100</monBatchSize>
  <monCompression>none</monCompression>
  <monBatchDuration>10</monBatchDuration>
</Monitor>
```

Example: Create a TCP monitor

The following sample shows how to create a TCP monitor.

```
POST /ws/Monitor
```

```
<Monitor>
   <monTopic>DeviceCore,XbeeCore</monTopic>
   <monTransportType>tcp</monTransportType>
   <monFormatType>json</monFormatType>
   <monBatchSize>100</monBatchSize>
   <monCompression>none</monCompression>
   <monBatchDuration>10</monBatchDuration>
   <monAckOption>explicit</monAckOption>
   <monAutoReplayOnConnect>true</monAutoReplayOnConnect>
</Monitor>
```

Example: Recover a disabled monitor

An HTTP monitor that is not able to successfully connect over a 24 hour period is disabled. Once disabled:

- System alarm is generated to indicate the monitor state was changed to disabled.
- Remote Manager does not make any more attempts to connect the monitor.
- Persistent monitors no longer store missed monitor events.

To recover a disabled monitor, re-enable the monitor using a PUT request. When recovered, the back-off sequence is restarted from the beginning. The minimum content required in the PUT is the monId element (21 in the following example):

PUT /ws/Monitor

```
<Monitor>
   <monId>21</monId>
</Monitor>
```

Example: Delete a monitor

The following sample shows how to delete a monitor.

```
DELETE /ws/Monitor/148214
```

Example: Delete monitors based on conditions

The following examples shows how to delete all TCP monitors that are currently inactive:

```
DELETE ws/Monitor?condition=monTransportType='tcp' and monStatus='INACTIVE'
```

Example: Create a polling monitor

The following sample shows how to create a polling monitor.

Request

```
POST /ws/Monitor
```

```
<Monitor>
  <monTopic>DeviceCore,DataPoint/00000000-00000000-00000000-00000000</monTopic>
  <monTransportType>polling</monTransportType>
  <monDescription>Query monitor saves push notifications but does not send
them.</monDescription>
</Monitor>
```

Response

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
  <location>Monitor/433016</location>
</result>
```

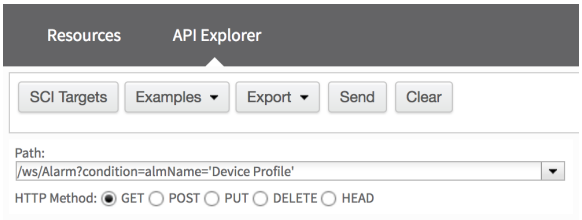
To query the data from a polling monitor, see v1/monitors/history.

Example: Monitor Profile Manager status with a push monitor

The following example shows how to update an external application using a push monitor to show the results of profile manager. This example consists of the following steps:

- Determine profile alarm ID
- Note the alarm ID in the <almid> tag

Issue http get to /ws/Alarm



View the results in the response pane

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
  <resultTotalRows>1</resultTotalRows>
```

```
  <requestedStartRow>0</requestedStartRow>
  <resultSize>1</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <Alarm>
    <almId>12163</almId>
    <cstId>2899</cstId>
    <almtId>1</almtId>
    <grpId>3560</grpId>
    <almName>Device Profile</almName>
    <almDescription>Device Profile Alarm</almDescription>
    <almEnabled>true</almEnabled>
    <almPriority>1</almPriority>
    <almScopeConfig>
      <almScopeConfig/>
    </almScopeConfig>
    <almRuleConfig>
      <almRuleConfig/>
    </almRuleConfig>
  </Alarm>
</result>
```

Note the alarm ID in the <almId> tag

```
 <almId>12163</almId>
```

HTTP/HTTPS transport protocol

This section highlights the details associated with an HTTPS or HTTP connection between the Remote Manager server and the customer web server. This is a high speed, transport over a HTTP connection. This transport requires that the customer has a publicly facing web server application. Remote Manager will be the HTTP client and will push any configured published events to the customer's web server. This transport uses basic authentication and therefore HTTPS is recommended. HTTP is available for debugging or troubleshooting.

To configure an HTTP monitor, specify http as the monTransportType setting. Additionally, specify monTransportUrl and monTransportToken options.

monTransportType: (Required) Sets the transport type, TCP or HTTP. For HTTP, set the transport type to **HTTP**.

monTransportUrl: (Required) Specifies the URL of the customer web server. The URL should be of the following form:

http[s]://customer.domain.com/application/path

monTransportToken: (Required) Specifies the credentials for basic authentication in the following format:

username:password

monTransportMethod: (Optional) Specifies the HTTP method to use to send data: PUT or POST. The default is PUT.

The following example shows how to create an HTTP monitor:

```
<Monitor>
   <monTopic>DeviceCore,XbeeCore</monTopic>
   <monTransportType>http</monTransportType>
   <monTransportUrl>your website url</monTransportUrl>
   <monTransportToken>username:password</monTransportToken>
   <monTransportMethod>PUT</monTransportMethod>
   <monFormatType>json</monFormatType>
   <monBatchSize>100</monBatchSize>
   <monCompression>none</monCompression>
   <monBatchDuration>10</monBatchDuration>
</Monitor>
```

Protocol

Once the HTTP monitor has been configured, the monitor will be activated and Remote Manager will connect to the customer's web server. Any matching events will be published to the specified URL using the supplied token credentials. Please note that if the monitor's URL or credentials are configured incorrectly or if the customer's web server is unreachable, Remote Manager will periodically attempt to connect to the web server for up to 24 hours. The monitor will be disabled after 24 hours without a successful connection.

Events are published using the configured monTransportMethod: PUT or POST. The default is an HTTP PUT operation. The standard HTTP headers of the published event include:

- Authorization: Basic…
- Content-Type: "application/xml;charset=UTF-8" or "application/json;charset=UTF-8"

- Content-Length: indicates how many bytes of payload data are in the message
- [Content-Encoding: deflate] - if present, indicates the monitor event data is compressed

Additionally, the following custom header fields will be set to describe the payload being delivered:

- Monitor-Protocol-Version: indicates what version of push protocol is being used. The current version is '1'.
- Monitor-DataBlockId: a rotating integer ID that identifies the data block.
- Monitor-Aggregate-Count: the number of publish events included in this batch.

The body of the PUT operation is the published event payload data. Its format, compression, and size are indicated in the headers above. The payload data format is the same as for the TCP transport.

The returned HTTP status code indicates the ability of the customer application to receive and process the data:

- 200 - indicates customer application successfully received and processed the data

Monitor published events payload

Data is encapsulated in a message envelope that includes the topic, operation, and timestamp plus the data itself. This will be formatted according to the format type requested when establishing the monitor. Additionally, when the monAutoReplayOnConnect option is enabled, there will be a replay="true" attribute if the message is being resent.

XML format

```
<?xml version="1.0" encoding="UTF-8"?>
<Msg topic="3/DeviceCore/882/7" operation="create|update|delete" timestamp="2010-
12-03T13:34:00.001Z" [replay="true"]>
    <DeviceCore>
        <id>
            <devId>882</devId>
            <devVersion>7</devVersion>
        </id>
        <devRecordStartDate>2010-12-03T13:34:00Z</devRecordStartDate>
        <devMac>00:40:9D:3D:71:15</devMac>
        <devConnectwareId>00000000-00000000-00409DFF-FF3D7115</devConnectwareId>
        ...
    </DeviceCore>
</Msg>
```

JSON format

```
{
   "Document":{
      "Msg":{
         "timestamp":"2010-12-03T13:34:00.001Z",
         "topic":"3/DeviceCore/882/7",
         "operation":"UPDATE",
         "DeviceCore":{
            "id":{
               "devId":882,
               "devVersion":7
            },
            "devMac":"00:40:9D:3D:71:15",
            "...": "..."
         }
      }
   }
}
```

TCP transport protocol

This section details standard TCP/IP and SSL socket connections between a client application and Remote Manager. Because authentication messages flow across the socket, we strongly recommend using SSL. Use standard TCP/IP connections for debugging and troubleshooting only.

Monitor configuration options for TCP

The Monitor API provides two TCP-specific elements:

monTransportType: (Required) Sets the transport type, TCP or HTTP. For TCP, set the transport type to **TCP**.

monAckOption: (Optional) Specifies acknowledge options for sent messages.

- **explicit**: Client must explicitly acknowledge TCP push events.
- **off**: Remote Manager automatically acknowledges events when sent.

The default is off.

The following example shows how to create a TCP monitor:

```
<Monitor>
   <monTopic>DeviceCore,XbeeCore</monTopic>
   <monTransportType>tcp</monTransportType>
   <monFormatType>json</monFormatType>
   <monBatchSize>100</monBatchSize>
   <monCompression>none</monCompression>
   <monBatchDuration>10</monBatchDuration>
   <monAckOption>explicit</monAckOption>
   <monAutoReplayOnConnect>true</monAutoReplayOnConnect>
</Monitor>
```

Protocol

When a monitor is created through the Web Services API, a Monitor ID is assigned and returned to the caller. If the monitor is configured to use the TCP transport the customer application can activate the monitor by establishing a TCP socket connection back to the Remote Managerserver. SSL monitor sockets should be made to port 3201 while unsecure TCP sockets should be made to port 3200.

Once Remote Manager makes the socket connection, the customer application must send a ConnectRequest message through that connection to the Remote Manager server. The server will authenticate the request and send back a response. Once the connect request succeeds, the server will begin sending PublishMessages to the customer application as events matching the monitor configuration occur. There are two options on how the customer application can acknowledge the PublishMessages: **explicit** and **off**. The acknowledgment option is configured using the monAckOption in the Monitor web service. If not specified, the monAckOption defaults to off.

Explicit means that the customer application will acknowledge the receipt of PublishMessages using the PublishMessageReceived message. The dataBlockId in the PublishMessageReceived indicates that all events up to and including that dataBlockId were successfully received, i.e. one PublishMessageReceive message can acknowledge multiple PublishMessages. If the customer application detects a missing dataBlockId or cannot process a PublishMessage, it should disconnect the TCP socket. On the next reconnect, the replay will start with the unacknowledged push event. (Note that monAutoReplayOnConnect needs to be enabled.)

The **off** option means that Remote Manager will treat the push event as acknowledged when it is written to the TCP socket. Any PublishMessageReceived messages will be ignored by Remote Manager if the monitor is configured with monAckOption set to off.

As long as the monitor socket connection remains open, monitor events will flow from the server to the customer application per the requirements established in the monitor configuration. If the socket is closed for any reason, the monitor will be deactivated and monitor events will stop flowing to the customer application. When the monitor is deactivated, the monitor's status will be marked as SUSPENDED (for monitors configured for auto replay of missed events using monAutoReplayOnConnect), otherwise INACTIVE. The customer application can reactivate the monitor socket in the same manner as the initial connection.

Conventions

In this protocol, all multi-byte numeric fields must be transmitted in big endian format. All text data must be transmitted as UTF-8 characters. See RFC 2279 as a reference for this format.

Framing

All messages between the client application and the Remote Manager server are framed as follows:

- Header [6 Bytes]
  - Type: [2 Bytes] - indicates the type of message being exchanged
  - Length: [4 Bytes] - indicating size of the framed message payload
- Payload [n Bytes] - the wrapped message

ConnectRequest message

To initiate a new monitor connection, send a ConnectRequest message from the client application to Remote Manager. This is the first message sent upon connect and will authenticate and activate the monitor.

Header [6 Bytes] Type=0x0001

Payload:

- ProtocolVersion: [2 Bytes] - indicates what version of push protocol is being used. The current version is 0x0001.
- UserNameLen [2 Bytes] - length of UserName payload
- UserName: [UTF-8 encoded byte array] - the username to authenticate connection
- PasswordLen [2 Bytes] - length of Password payload
- Password: [UTF-8 encoded byte array] - the password to authenticate connection
- MonitorId: [4 Bytes] - the ID of the monitor for this connect

Example

offset	bytes															
0x0000	type		size				p. version		userSize		username					si..
	00	01	00	00	00	13	00	01	00	05	70	65	70	73	69	00
0x0010	..ze	password				message id										
	04	63	6F	6C	61	00	00	01	04							
0x0020																

Legend:

Type: 0x0001
Size: 0x00000013
ProtocolVersion: 0x0001
UsernameSize: 0x0005
Username: 0x7065707369 (pepsi)
PasswordSize: 0x0004
Password: 0x636f6c61 (cola)
MessageId: 0x00000104

ConnectResponse message

The response to ConnectRequest, sent from Remote Manager to the client application, is a ConnectResponse message. This indicates to the client application the status of the web services request, as well as the protocol version that Remote Manager is speaking.

Header [6 Bytes] Type=0x0002

Payload:

- Status Code: [2 Bytes]
- ProtocolVersion: [2 Bytes] - indicates what version of push protocol is being used

Example:

offset	bytes															
0x0000	type		size				status		p. ver							
	02	00	00	00	04	00	01	00	01							
0x0010																

Legend:

Type: 0x0002
Size: 0x00000004
Status: 0x0001
ProtocolVersion: 0x0001

PublishMessage message

As monitored events occur, Remote Manager will send PublishMessage messages to the client application.

Header [6 Bytes] Type=0x0003
Payload:

- DataBlockId: [2 Bytes] - rotating id that uniquely identifies the data block
- Count: [2 Bytes] - number of messages in this batch
- Compression: [1 Byte] - indicates what payload compression algorithm is being used (0x00=none, 0x01=zlib)
- Format: [1 Byte] - indicates data format of payload (0x00=xml, 0x01=json)
- PayloadSize: [4 Bytes] - indicates how many bytes of payload data follow
- PayloadData: [n Bytes] - the actual Monitor event data (may be compressed & Base64 encoded)

Example:

offset	bytes															
0x0000	type		size				DataBlockId		count		cmp	fmt	payload size			
	00	03	00	00	02	15	01	A7	00	02	00	00	00	00	02	05
0x0010	payload data															
	3C	44	6F	63	75	6D	65	6E	74	3E	3C	4D	73	67	20	74
...	payload data															
	...															
0x0020	payload data															
	6D	65	6E	74	3E											

Legend:

Type: 0x0003
Size: 0x00000215
DataBlockId: 0x01A7
Count: 0x0002
Compression: 0x00
Format: 0x00
PayloadSize: 0x00000205
PayloadData: 0x3C446F63756D656E74 … 6E743E

```
<Document>
    <Msg topic="3/DeviceCore/882/7" operation="update" timestamp="2010-12-
03T13:34:00.001Z">
        <DeviceCore>...</DeviceCore>
    </Msg>
    <Msg
topic="3/XbeeCore/00:13:A2:00:40:01:60:45/1/0/1794/256"operation="update"
timestamp="2010-12-03T13:34:00.001Z">
        <XbeeCore>...</XbeeCore>
    </Msg>
</Document>
```

PublishMessageReceived message

In response to a PublishMessage message, the client application will send a PublishMessageReceived to acknowledge the message was received and what its processing status is.

Header [6 Bytes] Type=0x0004
Payload:

- DataBlockId: [2 Bytes] - corresponds to incoming DataBlockId
- Status: [2 Bytes] 200 - indicates customer application successfully received and processed the data

Example:

offset	bytes															
0x0000	type		size				DataBlockId		status							
	00	04	00	00	00	04	01	A7	00	C8						
0x0010																

Type: 0x0004
Size: 0x00000004
Status: 0x00C8

NetworkInterface

NetworkInterface contains specific information related to external network interfaces for devices where Remote Manager needs to have knowledge of that information in order to interact with those devices. For example, Remote Manager uses NetworkInterface records to associate phone numbers with one or more mobile identification numbers (SIM or modem serial number, depending upon the mobile technology).

URI

```
http://<hostname>/ws/NetworkInterface
```

Formats

HTTP Method	Format	Description
GET	/ws/NetworkInterface	Display a list of modems provisioned in your account.
POST	/ws/NetworkInterface	Add a modem to your account.
PUT	/ws/NetworkInterface	Update modem information for your account.
DELETE	/ws/NetworkInterface	Delete a modem from your account.

Elements

id

Element that uniquely identifies the network interface and consists of the following:

niId

System-generated identifier for the network interface.

niVersion

A value of 0 indicates the most recent version of the network interface record.

niPhoneCarrier

An integer that represents the telephone carrier service subscription for the network interface. The value is specific to the internal implementation of the SMS service used to send and receive SMS messages.

niTerminated

Boolean value that indicates whether the network interface is terminated. Enter either true or false.

niEffectiveStartDate

Date the network interface was added to Remote Manager.

cstId

Remote Manager identifier for the customer.

grpId

Remote Manager identifier for the customer group.

devId

Device ID of the device associated with this network interface record.

devVersion

A value of 0 which indicates the most recent version of the device record.

niInterfaceType

Integer that indicates the network interface type:

0: None

1: GSM

2: CDMA

3: ORBCOMM

4: Iridium

niSimId

Network interface SIM indentifier which is the ICCID, MEID, or ESN of the SIM or cellular modem.

niModemId

Modem ID of the satellite modem.

niPhone

Telephone number of the cellular line using international format for telephone numbers. For example:

+1 123-456-7890

niActivePhone

Boolean value that indicates whether this network interface record contains the telephone number (niPhone) to which Remote Manager sends SMS messages for this device. Only one NetworkInterface record can have niActivePhone set to true per device.

niIdigiPhone

Short or long code the device uses to communicate with Remote Manager.

niIdigiServiceId

Keyword used in the shared code

niImsi

International Mobile Subscriber Identity (IMSI) of the SIM.

NetworkInterfaceSubscriptionCore

Use the NetworkInterfaceSubscriptionCore web service to get subscription information for your devices based on the network interface records. The listing indicates whether carrier accounts have been assigned to the network interface records and what metrics are collected for each.

URI

```
http://<hostname>/ws/NetworkInterfaceSubscriptionCore
```

Formats

HTTP Method	Format	Description
GET	/ws/NetworkInterfaceSubscriptionCore	Display a list of modems provisioned in your account.

Elements

cstId

Remote Manager identifier for the customer.

id

Element that uniquely identifies the network interface and consists of the following:

niId

System-generated identifier for the network interface.

subId

Subscription identifier.

Remote command interface (RCI)

Remote Command Interface (RCI) is a mechanism that allows remote configuration, control, and information exchange between an RCI client, typically a web services client acting via Remote Manager, and an RCI target, typically a Digi device implementing the RCI specification.

RCI consists of a transport mechanism, such as the Remote Manager device protocol, EDP, and an XML-based request and reply document specification. For complete information on RCI, see Remote Command Interface (RCI) specification.

Schedule

Use the Schedule web service to create, modify, or delete a schedule.

URI

```
http: //<hostname>/ws/Schedule
```

Formats

HTTP method	Format	Description
GET	/ws/Schedule	Get a list of scheduled tasks for your account.
POST	/ws/Schedule	Create a schedule for a defined task template.
PUT	/ws/Schedule/{schId}	Modify a schedule.
DELETE	/ws/Schedule/{schId}	Delete a schedule.

Elements

schId

System-generated identifier for the schedule.

schDescription

Description of the schedule.

schExpression

Expression that determines when the schedule runs: For example, IMMEDIATE.

schTargets

One or more targets for the schedule tasks.

schStartTime

Time at which the schedule started execution.

schStopTime

Time at which the schedule stopped execution.

schStatus

Current status of the schedule:

0 = new

1 = in_progress

3 = complete

4 = canceled

schPreviousRunTime

Time at which the schedule was last executed.

task

Task template associated with the schedule.

Example: Schedule device reboot

The following example shows how to schedule a device to reboot immediately. The example uses the Schedule and Task APIs to post the schedule to your device and then steps through the process of verifying that your device rebooted successfully.

Request

```
GET /ws/Task
```

Response

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
  <resultTotalRows>1</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>1</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <Task>
    <tskId>732</tskId>
    <schId>258</schId>
    <cstId>2</cstId>
    <usrId>9</usrId>
    <tskScheduledTime>2014-12-14T20:59:58.070Z</tskScheduledTime>
    <tskStartTime>2014-12-14T20:59:58.200Z</tskStartTime>
    <tskTargets>00000000-00000000-886123FF-FF000026</tskTargets>
    <tskSuccess>0</tskSuccess>
    <tskFailures>0</tskFailures>
    <tskStatus>1</tskStatus>
    <tskRequestPayload>
      <description>Another Schedule</description>
```

```
      <command>
        <name>List Files</name>
        <event>
          <on_error>
            <end_task/>
          </on_error>
        </event>
        <sci>
          <file_system allowOffline="true">
            <commands>
              <ls path="/"/>
            </commands>
          </file_system>
        </sci>
      </command>
    </tskRequestPayload>
    <tskTargetCount>1</tskTargetCount>
    <tskDescription>Another Schedule</tskDescription>
  </Task>
</result>
```

SCI (Server command interface)

SCI (Server Command Interface) is a web service that allows users to access information and perform commands that relate to their device. SCI is available only in Premier Edition accounts. If you are unable to use SCI, contact Remote Manager support to make changes to your account.

Examples of SCI requests include:

- Retrieve live or cached information about your device(s)
- Change settings on your device(s)
- Interact with a Python program running on your device(s) to send commands or retrieve information
- Read from and write to the file system on your device(s)
  - Update your Python applications
  - Retrieve data stored locally on your device(s)
- Update device firmware
- Update XBee radio firmware on your device(s)
- Remotely reboot your device(s)

The operations can be performed synchronously or asynchronously. Synchronous requests are useful if you would like to execute a short request to the server and block until the operation is completed. Asynchronous requests are useful when you want to execute a request that has the possibility of taking a while to complete or you simply want to send the request off and return immediately. With asynchronous requests, you receive an ID that you can later use to check on the job status and retrieve results.

An SCI request is composed of XML that is posted to http(s)://`<hostname>`/ws/sci.

SCI request

Every SCI request looks like the following:

```
<sci_request version="1.0">
    <{operation_name}>
        <targets>
          {targets}
        </targets>
          {payload}
    </{operation_name}>
</sci_request>
```

operation_name is either send_message, update_firmware, disconnect, or query_firmware_targets
targets contains one or more elements that look like: `<device id="{device_id}"/>`, `<device id="all"/>`, `<device tag="{tag name}"/>`, or `<group path="{group name}"/>`
payload is dependent on the operation

File Reference

The payload for an SCI command can be referenced from a file in Remote Manager Data Services as opposed to being explicitly described in the actual request. For example, the following SCI request payload is referenced instead of explicitly declared in the XML:

```
<sci_request version="1.0">
    <send_message>
        <targets>
            <device id="00000000-00000000-00000000-00000000"/>
        </targets>
        <file>/~/my_commands/set_settings.xml</file>
    </send_message>
</sci_request>
```

Where the content of set_settings.xml could be similar to the following:

```
<rci_request>
   <set_setting>
      <query_setting>....</query_setting>
   </set_setting>
</rci_request>
```

SCI targets

The targets field within an SCI request can be one of the following elements:

- `<device id="{device_id}"/>`
  When included in an SCI request, this element specifies a particular device ID. Requests issued will only be sent to the specified device.
- `<device id="all"/>`
  When included in an SCI request, this element specifies the device IDs of all of your Remote Manager-registered devices. Requests issued will be sent to all of the devices registered within your Remote Manager user account.
- `<device tag="{tag name}"/>`
  When included in an SCI request, this element specifies a particular tag name. Requests issued will be sent to all of the devices containing the specified tag name.
- `<group path="{group name}"/>`
  When included in an SCI request, this element specifies a particular group name. Requests issued will be sent to each of the devices contained within the specified group.

Note Each element under Targets can be thought of as an OR statement, thus you can specify multiple group paths, and it will effect each path specified.

Synchronous requests

To send a synchronous request using a device ID:

POST the following to: /ws/sci

```
<!-- common to every sci request -->
<sci_request version="1.0">
    <!-- indicates we want to send an rci request -->
   <send_message>
      <!-- preparing us for the list of who to operate on -->
      <targets>
         <!-- we will send it to this device -->
         <device id="00000000-00000000-00000000-00000000" />
      </targets>
      <!-- the payload for the send_message command, an rci request -->
```

```
      <rci_request version="1.1">
         <query_state>
            <device_stats />
         </query_state>
      </rci_request>
   </send_message>
</sci_request>
```

which returns when the operation has completed (or timed out) and the body of the response will be:

```
<sci_reply version="1.0">
<!-- start of the sci response -->
<send_message>
  <!-- the "operation" of our sci_request -->
   <device id="00000000-00000000-00000000-00000000">
      <!-- contains the response for this device -->
      <rci_reply version="1.1">
          <!-- the rci response for the particular device -->
         <query_state>
            <device_stats>
               <cpu>36</cpu>
               <uptime>152</uptime>
               <datetime>Thu Jan 1 00:02:32 1970 (based on uptime)</datetime>
               <totalmem>8388608</totalmem>
               <usedmem>5811772</usedmem>
               <freemem>2576836</freemem>
            </device_stats>
         </query_state>
      </rci_reply>
   </device>
</send_message>
```

To send a synchronous request using a group path:

POST the following to /ws/sci

```
<!-- common to every sci request -->
    <sci_request version="1.0">
       <!-- indicates we want to send an rci request -->
       <send_message>
          <!-- preparing us for the list of who to operate on -->
          <targets>
             <!-- we will send it to this group -->
             <group path="group1" />
          </targets>
          <!-- the payload for the send_message command, an rci request -->
          <rci_request version="1.1">
             <query_state>
                <device_stats />
             </query_state>
          </rci_request>
       </send_message>
    </sci_request>
```

which will return when the operation has completed (or timed out) and the body of the response will be:

Note The return will contain a response for each device included within the specified group.

```
   <sci_reply version="1.0">
    <!-- start of the sci response -->
    <send_message>
      <!-- the "operation" of our sci_request -->
       <device id="00000000-00000000-00000000-00000001">
          <!-- contains the response for the first device in the specified group
-->
          <rci_reply version="1.1">
            <!-- the rci response for the first device in the specified group -->
             <query_state>
                <device_stats>
                   <cpu>22</cpu>
                   <uptime>237565</uptime>
                   <totalmem>8388608</totalmem>
                   <usedmem>7136532</usedmem>
                   <freemem>1252076</freemem>
                </device_stats>
             </query_state>
          </rci_reply>
       </device>
       <device id="00000000-00000000-00000000-00000002">
          <!-- contains the response for the second device in the specified group
-->
          <rci_reply version="1.1">
            <!-- the rci response for the second device in the specified group --
>
             <query_state>
                <device_stats>
                   <cpu>42</cpu>
                   <uptime>438054</uptime>
                   <datetime>Mon Jun 28 19:36:29 2010</datetime>
                   <totalmem>8388608</totalmem>
                   <usedmem>8165060</usedmem>
                   <freemem>223548</freemem>
                </device_stats>
             </query_state>
          </rci_reply>
       </device>
    </send_message>
    <sci_request>
```

To send a synchronous request using a device tag:

POST the following to: http://remotemanager.digi.com/ws/sci

```
<!-- common to every sci request -->
<sci_request version="1.0">
 <!-- indicates we want to send an rci request -->
<send_message>
   <!-- preparing us for the list of who to operate on -->
   <targets>
      <!-- we will send it all devices that have this tag -->
      <device tag="tag1" />
   </targets>
   <!-- the payload for the send_message command, an rci request -->
   <rci_request version="1.1">
      <query_state>
         <device_stats />
      </query_state>
   </rci_request>
```

```
</send_message>
</sci_request>
```

which will return when the operation has completed (or timed out) containing responses from all of the devices matching the specified tag.

Asynchronous request

SCI requests that are asynchronous return without waiting for the request to finish, and return a job ID that can be used to retrieve the status and results later.

If you POST an SCI request asynchronously and want to see the results, the general flow is:

POST the SCI request.

```
if rc=202 // The job is accepted
      get the location from the response header or the job ID from the response
content
      rc = HEAD location
      while rc!=200
            sleep for a number of seconds
            rc = HEAD location
      GET location
      process your results
      DELETE location
```

Performing an Asynchronous Request

A synchronous request is performed by specifying *synchronous="false"* in the element specifying the operation in the request, e.g.: `<send_message synchronous="false">`

the response then has the form:

```
<sci_reply version="1.0">
   <send_message>
      <jobId>{job_id}</jobId>
   </send_message>
</sci_reply>
```

where job_id identifies the request you submitted.

Retrieve Status

You can retrieve the status for a particular request, or retrieve information about submitted requests overall.

Status for a Particular Job

Do an HTTP GET on http://remotemanager.digi.com/ws/sci/{job_id}

To determine if a job is complete, do an HTTP HEAD specifying the job ID; http://remotemanager.digi.com/ws/sci/601358. A return code of 200 means the job is complete.

Overall Status of Outstanding Jobs

Do an HTTP GET on /ws/sci, and you will get a response that looks like:

```
<result>
   <resultTotalRows>1</resultTotalRows>
   <requestedStartRow>0</requestedStartRow>
```

```
   <resultSize>1</resultSize>
   <requestedSize>1000</requestedSize>
   <remainingSize>0</remainingSize>
   <Job>
      <jobId>601358</jobId>
      <cstId>0</cstId>
      <usrId>0</usrId>
      <jobType>0</jobType>
      <jobSyncMode>0</jobSyncMode>
      <jobReplyMode>0</jobReplyMode>
      <jobTargets>00000000-00000000-0004F3FF-00000000</jobTargets>
      <jobRequestPayload>&lt;rci_request&gt;&lt;query_setting/&gt;&lt;/rci_
request&gt;</jobRequestPayload>
      <jobDescription>query_setting</jobDescription>
      <jobStatus>2</jobStatus>
      <jobTargetCount>1</jobTargetCount>
      <jobProgressSuccess>1</jobProgressSuccess>
      <jobProgressFailures>0</jobProgressFailures>
      <jobSubmitTime>2010-03-02T15:36:22Z</jobSubmitTime>
      <jobCompletionTime>2010-03-02T15:36:22Z</jobCompletionTime>
   </Job>
</result>
```

where **jobId** is the ID for the request.
jobType is the type of the job (0: send_message, 1: update_firmware, 2: disconnect).
jobSyncMode indicates if the job is synchronous (0: synchronous, 1: asynchronous).
jobReplyMode indicates the reply mode (0: all, 1: none, 2: only), where only means only return errors.
jobStatus is the current status of the job (0: new, 1: in_progress, 2: complete, 3: canceled).
jobTargetCount is the number of devices the job is targeted for.
jobProgressSuccess is the number of devices that have completed the operation successfully.
jobProgressFailures is the number of devices that have completed the operation with an error.

Retrieve Progress

You can retrieve the progress for a particular SCI job by specifying the progress=true parameter. For example:

```
HTTP GET https://remotemanager.digi.com/ws/sci/{job_id}?progress=true
```

This will return the current progress (percent completion) for an SCI job, as well as its progress history. For example, let's assume we have an SCI job that is performing a firmware update on two different devices. Performing the query shown above will give a response that looks something like:

```
<sci_reply version="1.0">
   <status>in_progress</status>
   <update_firmware>
      <device id="00000000-00000000-000000FF- FF000001">
         <progress time="Mon Nov 28 21:30:25 UTC 2011" status="0">Getting Target
Info</progress>
         <progress time="Mon Nov 28 21:30:27 UTC 2011" status="0">Sending
Download Request</progress>
         <progress time="Mon Nov 28 21:31:15 UTC 2011" status="5">Sending Data:
156672 out of 3130662 bytes sent</progress>
         <progress time="Mon Nov 28 21:32:07 UTC 2011" status="10">Sending Data:
313344 out of 3130662 bytes sent</progress>
      </device>
      <device id="00000000-00000000-000000FF- FF000002">
         <progress time="Mon Nov 28 21:30:26 UTC 2011" status="0">Getting Target
```

```
Info</progress>
         <progress time="Mon Nov 28 21:30:27 UTC 2011" status="0">Sending
Download Request</progress>
         <progress time="Mon Nov 28 21:31:05 UTC 2011" status="5">Sending Data:
156672 out of 3130662 bytes sent</progress>
         <progress time="Mon Nov 28 21:31:48 UTC 2011" status="10">Sending Data:
313344 out of 3130662 bytes sent</progress>
         <progress time="Mon Nov 28 21:32:30 UTC 2011" status="15">Sending Data:
470016 out of 3130662 bytes sent</progress>
      </device>
   </update_firmware>
</sci_reply>
```

We can also query for job progress on other types of SCI jobs, including file uploads through the File System service. Progress updates for file uploads through RCI is not supported.

Cancel a Request or Delete the Results

Do an HTTP DELETE on https://remotemanager.digi.com/ws/sci/{job id}

This will attempt to cancel the request. Some parts of the request may have already completed, and parts of the request that are in progress may continue to completion, but it should prevent any operations that have not yet begun from starting.

Ping request

You can use the ping command to determine the round trip latency of a device connection. The result gives the actual time used to send a simple command to the device and receive a reply.

You can ping a device using the following SCI request.

```
<sci_request version="1.0">
    <ping>
       <targets>
          <device id="00000000-00000000-00042DFF-FF04A85A"/>
       </targets>
    </ping>
</sci_request>
```

The sample response contains the actual time used to send a simple command to the device and receive a reply.

```
<sci_reply version="1.0">
    <ping>
       <device>
             <device id="00000000-00000000-00042DFF-FF04A85A">
             <time>
               <device units="ms">5</device>
             </time>
       </device>
    </ping>
</sci_reply>
```

Available operators

Available operators include:

send_message allows an RCI request to be sent to the device (or the server cache).

update_firmware updates the firmware of the device.

disconnect sends a request to the device to disconnect from the server.

query_firmware_targets gets a list of firmware targets on the device.

file_system is used to interact with files on a device. This interface is for use with devices supporting the file_system service (as opposed to other devices that support file_system interaction through RCI requests.) The RCI do_command for file_system is only supported by older devices not implementing the file_system service.

data_service sends messages to devices over the data service.

reboot issues a reboot for a device.

There are a few attributes that can be specified for an operation that can specify the behavior. They include:

```
<{operation_name} reply="all|errors|none">
<{operation_name} synchronous="true|false">
<{operation_name} syncTimeout="xxx">
<{operation_name} cache="true|false|only">
<{operation_name} allowOffline="true|false">
<{operation_name} waitForReconnect="true|false">
```

reply determines how much information should be saved in the response to a request.

all (default) means that everything should be saved.

errors implies that only errors in the response should be kept, while success messages should not be saved.

none means that result data for the operation should not be saved.

errors is useful if you are performing an operation and only want to see error information that occurred, such as when setting settings, or performing firmware update. **none** is useful when you aren't concerned with the reply at all. If you are performing a synchronous request because you want to wait until the operation is complete, but do not want to receive a lot of data in the reply, this would accomplish that.

synchronous determines whether the request should be sent synchronously (default), or asynchronously (false).

syncTimeout is applicable for a synchronous request and determines how long to wait for the request to complete (in seconds) before an error is returned. The default changes based on the type. The overall timeout for the SCI request will be the accumulation of all operations combined. Unless overridden with syncTimeout, the timeouts on send_message commands are as follows:

operation	timeout
default	1 min
do_command target="file_system"	10 min
do_command target="zigbee"	5 min
firmware update	5 min

cache determines if the request should be processed on the server if possible, or always sent to the device. Options include:

- *true* (default) means that if possible, the request will be processed from the server cache without being sent to the device. If it cannot, it will be sent to the device.
- *false* means that the request will bypass the server cache and be sent to the device.
- *only* means that the request should only be processed by the server cache and will never be sent to the device, even if the server is not capable of servicing the request.

allowOffline determines if this should be an offline operation. Offline operations enable you to send a request to a device that is currently disconnected. If the device is already connected, then Remote Manager will execute the command right away. If the device is not connected, then Remote Manager will execute the command as soon as the device connects to Remote Manager. Offline requests can be specified by setting the allowOffline attribute to "true".

NOTES:

- By default, SCI requests are synchronous. For offline operations, it is recommended to use an asynchronous request by setting the synchronous attribute to "false".
- Asynchronous offline operations will timeout after 7 days.
- If for some reason the device disconnects during processing, the operation will automatically be retried again the next time the device connects. Offline operations will be retried up to 5 times before failing.

waitForReconnect allows the completion of a command to depend on a device reconnecting. For example, normally sending a reboot command to a device would result in the command being marked as successfully completed as soon as the device acknowledges the reboot request. However, in many instances, it may make sense to wait to mark the command as successful until the device reconnects to Remote Manager. In such cases, this can be achieved by setting the waitForReconnect attribute to "true".

Warning: Many commands do not result in the device disconnecting and reconnecting to Remote Manager, meaning that improper use of this setting could result in the request taking an indefinite amount of time to complete; use caution when using this setting.

send_message

This is used to send an RCI request to a device. The reply will contain the response from the devices or groups of devices, or any error messages. A device ID of all will cause the RCI request to be sent to all devices available to the user.

One of the main uses of RCI requests are to interact with the settings and state of a device. Remote Manager keeps a cache of the latest settings and state that it has received from a device, and this makes it possible to retrieve information about the state or settings of a device without having to go to the device.

The format of the send_message command is as follows:

```
<sci_request version="1.0">
   <send_message>
      <targets>{targets}</targets>
      <rci_request version="1.1">
         <!-- actual rci request -->
      </rci_request>
   </send_message>
</sci_request>
```

query_setting

Request for device settings. May contain setting group elements to subset query (only setting group subset supported. Subsetting below this level not supported).

Returns requested config settings. Requests specifying no settings groups (e.g. return all settings).

set_setting

Set settings specified in setting element. Settings data required.

Empty setting groups in reply indicate success. Errors returned as error or warning elements.

query_state

Request current device state such as statistics and status. Sub-element may be supplied to subset results.

Returns requested state. Requests specifying no groups (e.g. return all state).

Example:

```
<query_state>
  <device_stats/>
</query_state>
```

reboot

Reboots device immediately.

do_command

The do_command is a child element of an RCI Request that is executed differently based on the value of its target attribute. Note that the command may not be supported for use with your device. See the file_system service.

Python

You can add callbacks to unhandled do_commands target via the rci python module on a device.

File System The file_system commands are accessed via the do_command of an rci request.

ls

Reports all files in a given directory.

Attributes:

- **dir** the path in which to list available files

Example:

```
<rci_request version="1.1">
 <do_command target="file_system">
   <ls dir="/WEB/python"/>
 </do_command>
</rci_request>
```

returns

```
<rci_reply version="1.1">
<do_command target="file_system">
 <ls dir="/WEB/python">
  <file name="python.zip" size="144321"/>
  <file name="digi_daq.zip" size="458980"/>
  <file name="digi_daq.yml" size="5270"/>
  <file name="digi_daq.py" size="6387"/>
  <file name="zigbee.py" size="1147"/>
 </ls>
</do_command>
</rci_reply>
```

get_file

Returns the base 64 encoded raw data from a file in the file system denoted by the name attribute.

Attributes:

- **name** path and destination filename for the file

Example:

```
<rci_request version="1.1">
 <do_command target="file_system">
   <get_file name="/WEB/python/python.zip"/>
 </do_command>
</rci_request>
```

returns

```
<rci_reply version="1.1">
    <do_command target="file_system">
        <get_file name="/WEB/python/python.zip">
           <data>UEs...KYmwaR</data>
        </get_file>
    </do_command>
</rci_reply>
```

put_file

Uploads a base 64 encoded file onto the device.

Attributes:

- **name** path and destination filename for the file

Example:

```
<put_file name="/WEB/destination.txt">
    <data>BASE64DATA</data>
</put_file>
```

rm

Removes a given file.

Attributes:

- **name** path and destination filename for the file

Example:

```
<rm name="/WEB/python/somefile.py"/>
```

Zigbee

The zigbee rci command interacts with an xbee module.

ZigBee Discover

Returns back a list of discovered nodes, with the first indexed node being the node in the gateway.

Optional attributes:

- **start** says the rci should return the nodes whose index is >= start. For some reason, if start > 1, the Gateway will return this list from cache, and not perform an actual discovery.
- **size** Determines number of nodes to return

- **clear** If this is set to "clear", it forces a clearing of the device's cache, and will always perform a discover to get fresh results

Example:

```
<do_command target="zigbee">
  <discover start="1" size="10" option="clear"/>
</do_command>
```

ZigBee Query Setting

Returns back a detailed list of settings for a given radio

Optional attribute:

- **addr** 64 bit address of the node to retrieve settings for. If omitted, defaults to gateway node

Example:

```
<do_command target="zigbee">
  <query_setting addr="00:13:a2:00:40:34:0c:88!"/>
</do_command>
```

ZigBee Query State

This is identical to query_setting, except it returns back different fields.

ZigBee Set Setting

Basically the reverse of query_setting, so you can set settings for a particular node

Optional attributes:

- **addr** 64 bit address of node to set settings for. If omitted, defaults to gateway node

Example:

```
<do_command target="zigbee">
<set_setting addr="00:13:a2:00:40:34:0c:88!">
     <radio>
         <field1>value1</field1>
         ...
         <fieldn>valuen</fieldn>
     </radio>
</set_setting>
</do_command>
```

ZigBee Firmware Update

Updates the firmware of the radio in the gateway

Required attribute:

- **file** Path to a firmware file which must already exist on the gateway

Example:

```
<do_command target="zigbee">
  <fw_update file="/WEB/firmware_file"/>
</do_command>
```

update_firmware

This is used to update the firmware of one or more devices. The firmware image can be hosted in your Data Services directory on Remote Manager, or can be Base64 encoded and placed in a data element

within the update firmware command. The response marks each device as either submitted or failed. A response of "Submitted" means the process to send the firmware and update request to Remote Manager completed successfully.

It is still possible for the process to fail between Remote Manager and the device. You will need to go back and verify that the device firmware version has actually changed. You can do this by using the DeviceCore request defined earlier. You may also use the RCI command "query_state".

There are optional attributes filename, and firmware_target, which are included with the update_ firmware element.

filename needs to be specified if your target device supports multiple targets that can be updated in order to choose which to upgrade. These will match patterns specified by the device which can be discovered using the query_firmware_targets command.

firmware_target can be used to bypass the filename matching and force an upgrade on a particular target.

A request using a Base64 encoded file looks like:

```
<sci_request version="1.0">
   <update_firmware filename="abcd.bin">
      <targets>{targets}</targets>
      <data>{base64 encoded firmware image}</data>
   </update_firmware>
</sci_request>
```

and the reply looks like:

```
<sci_reply version="1.0">
   <update_firmware>
      <device id="00000000-00000000-00000000-000000">
         <submitted />
      </device>
   </update_firmware>
</sci_reply>
```

To do the update operation with a file stored in Remote Manager Data Services, use the `<file>` attribute:

```
<sci_request version="1.0">
    <update_firmware filename="abcd.bin">
        <targets>
            {targets}
        </targets>
        <file>~/firmware/abcd.bin</file>
    </update_firmware>
</sci_request>
```

The above example assumes that you created a directory called "firmware" off the root of your home directory in Data Services. "~" is an alias to the home directory of your account.

disconnect

Disconnect is used to indicate that a device should disconnect from the server. Based on the device's configuration, it will likely reconnect.

A request follows this format:

```
<sci_request version="1.0">
   <disconnect>
      <targets>{targets}</targets>
```

```
   </disconnect>
</sci_request>
```

and a response looks like:

```
<sci_reply version="1.0">
   <disconnect>
      <device id="00000000-00000000-00000000-00000000">
         <disconnected />
      </device>
   </disconnect>
</sci_reply>
```

query_firmware_targets

Query Firmware Targets is used to retrieve information about the upgradeable firmware targets of a device. It will return the target number, name, version, and code size. A pattern may also be returned in the response which indicates a regular expression that is used to determine the appropriate target for a given filename.

A request follows this format:

```
<sci_request version="1.0">
   <query_firmware_targets>
      <targets>{targets}</targets>
   </query_firmware_targets>
</sci_request>
```

and a response looks like:

```
<sci_reply version="1.0">
   <query_firmware_targets>
      <device id="00000000-00000000-00000000-00000000">
         <targets>
            <target number="0">
               <name>Firmware Image</name>
               <pattern>image\.bin</pattern>
               <version>7.5.0.11</version>
               <code_size>2162688</code_size>
            </target>
            <target number="1">
               <name>Bootloader</name>
               <pattern>rom\.bin</pattern>
               <version>0.0.7.5</version>
               <code_size>65536</code_size>
            </target>
            <target number="2">
               <name>Backup Image</name>
               <pattern>backup\.bin</pattern>
               <version>7.5.0.11</version>
               <code_size>1638400</code_size>
            </target>
         </targets>
      </device>
   </query_firmware_targets>
</sci_reply>
```

file_system

The file system command is used to interact with files on a device. This interface is for use with devices supporting the file system service (as opposed to other devices which support file system interaction through RCI requests).

Commands have the following general format:

```
<sci_request version="1.0">
   <file_system>
      <targets>{targets}</targets>
      <commands>{one or more file_system commands}</commands>
   </file_system>
</sci_request>
```

Support file system commands are as follows:

put_file

The put_file command is used to push new files to a device, or optionally write chunks to an existing file.

- **path** is a required attribute giving the file to write to.
- **offset** is an optional attribute specifying the position in an existing file to start writing at.
- **truncate** is an optional attribute indicating the file should be truncated to the last position of this put.

The payload is specified in one of two ways:

1. Child element data with the payload Base64 encoded
2. Child element file with a path to a file in storage to send

Example:
A put file operation using a file on the server as the source for the data. The contents will be inserted into the file /path_to/write1.ext, as offset 200. It is set to not truncate the file if it extends beyond the length of written data.

```
<sci_request version="1.0">
   <file_system>
      <targets>
         <device id="00000000-00000000-00000000-00000000" />
      </targets>
      <commands>
         <put_file path="/path_to/write1.ext" offset="200" truncate="false">
            <file>~/referencedfilename.xml</file>
         </put_file>
      </commands>
   </file_system>
</sci_request>
```

A put file with the data Base64 encoded and embedded in the request under the data element. Offset and truncate are not specified, so this example will create a new file if one does not exist, or overwrite an existing one.

```
<sci_request version="1.0">
   <file_system>
      <targets>
         <device id="00000000-00000000-00000000-00000000" />
      </targets>
```

```
      <commands>
         <put_file path="/path_to/write2.ext">
            <data>ZmlsZSBjb250ZW50cw==</data>
         </put_file>
      </commands>
   </file_system>
</sci_request>
```

get_file

The get_file command is used to retrieve a file from the device, either in its entirety or in chunks. There is currently a restriction such that the maximum retrieval size is 512KB. As a result, files greater than this size will have to be retrieved in chunks.

- **path** is a required attribute giving the file to retrieve.
- **offset** is an optional attribute specifying the position to start retrieving from.
- **length** is an optional attribute indicating the length of the chunk to retrieve.

Example:
The get file in this example will retrieve 64 bytes starting at offset 100 from the file /path_to/file.ext. Leaving off offset and length would cause the full file to be retrieved.

```
<sci_request version="1.0">
   <file_system>
      <targets>
         <device id="00000000-00000000-00000000-00000000" />
      </targets>
      <commands>
         <get_file path="/path_to/file.ext" offset="100" length="64" />
      </commands>
   </file_system>
</sci_request>
```

ls

The ls command is used to retrieve file listings and details.

- **path** is a required attribute specifying where to get file details for.
- **hash** is an optional attribute which indicates a hash over the file contents should be retrieved. Values include none, any, md5, sha-512, and sha3-512. Use **any** to indicate the device should choose its best available hash. (If you specify md5, sha-512, or sha3-512, the device may not support the hash or any hash mechanism).

Example:
This ls request will return a listing of the contents of /path_to_list. By specifying *hash="any"*, the response will include the most optimal hash available, if any. Leaving off the hash attribute will default it to none.

```
<sci_request version="1.0">
   <file_system>
      <targets>
         <device id="00000000-00000000-00000000-00000000" />
      </targets>
      <commands>
         <ls path="/path_to_list" hash="any" />
      </commands>
```

```
   </file_system>
</sci_request>
```

rm

The rm command is used to remove files.

- **path** is a required attribute specifying the location to remove.

Example:
This rm request will attempt to delete /path_to/file.ext

```
<sci_request version="1.0">
   <file_system>
      <targets>
         <device id="00000000-00000000-00000000-00000000" />
      </targets>
      <commands>
         <rm path="/path_to/file.ext" />
      </commands>
   </file_system>
</sci_request>
```

data_service

A new subcommand in SCI is now supported to send messages to a device over the data service. The command element is data_service, and it must contain an element requests. The requests element contains a device_request element, with required attribute target_name and optional attribute format. target_name specifies the data service target the request will be sent to. The text data contained in the device_request element is used as the payload for the request. If format is not specified, the content will be sent as text. If *format="base64"* is specified, the content will be Base64 decoded and sent to the target as a binary payload.

Example of text payload

```
<data_service>
   <targets>
      <device id="00000000-00000000-00000000-00000000" />
   </targets>
   <requests>
      <device_request target_name="myTarget">my payload string</device_request>
   </requests>
</data_service>
```

Example of binary payload

```
<data_service>
   <targets>
      <device id="00000000-00000000-00000000-00000000" />
   </targets>
   <requests>
      <device_request target_name="myBinaryTarget"
format="base64">YmluYXJ5ZGF0YS4uLg==</device_request>
   </requests>
</data_service>
```

reboot

Reboot is used to issue a reboot for a device. The majority of devices may not support this operation, and will instead support reboot through RCI. This option exists as an alternative for devices that may

not support RCI.

Example:

```
<reboot>
   <targets>
      <device id="00000000-00000000-00000000-00000000" />
   </targets>
</reboot>
```

SMS messages

There are two types of SMS data:

- **Data:** Data SMS messages are sent from the device using the python function **idigisms_send()** and the messages are stored in **FileData** (see FileData). Data from the device is stored as a file in storage:
  - Python programs specify the data contents and optionally the file name, called the path in **idigisms_send()**.
  - If a path is not specified, the file is stored with the name **SmsUnnamed**. If the **fdArchive** option is true, the file is archived.
- **DIA data:** Remote Manager parses DIA messages and adds the parsed messages to the DIA data specific storage or as a file in generic storage depending on whether the device has a DIA Data Management service subscription (see DIA (device integration application)).

Sending SMS messages using web services

Remote Manager sends SMS requests to registered devices using the SCI (Server Command Interface) web service, and the messages are handled in a manner similar to RCI (Remote Command Interface) messages.

- For more information on SCI, see SCI (Server command interface).
- For more information on RCI, see Remote Command Interface (RCI) specification.

To send an SMS message using the Remote Manager SCI web service, specify the message as a child of the of the SCI **<send_message>** operation using the **<sms>** tag as a child element. Remote Manager creates a job for each web services request, and the job can be synchronous or asynchronous. You can retrieve Remote Manager job results the same way as you retrieve results for RCI jobs.

Send message options and SMS messages

Use the following **<send_message>** options to control how Remote Manager handles SMS message requests:

- **synchTimeout = "xxx"**
  Behavior is identical to existing SCI requests.
- **reply = "all | errors | none"**
  Controls whether a reply is sent by the device back to the server for a command. Using this option, you can control the number of Remote Manager SMS messages directly sent. The default is **none**. Note that **all** and **errors** work the same way for all SCI requests, including SMS message requests.
- **cache = "true | false | only"**
  Remote Manager does not currently service SMS requests from the cache. Therefore,

specifying **only** returns an error. In addition, **true** or **false** result in the request being sent to the device. The default is false.

SMS command children

Command	Description
ping	Ping the device via SMS to determine whether device can receive SMS messages. No parameters.
provision	Configure the device with the Remote Manager server telephone number and optionally the service ID. The **restrict sender** flag must be off in the device for this command. No parameters.
raw	Send raw SMS message from the device with no modification from Remote Manager. This is useful in cases when customers wish to use every byte of the SMS message (Remote Manager protocol takes approximately 5 bytes per message of overhead) or when using a device that doesn't have Remote Manager protocol support but does have SMS support. SMS raw messages can be a maximum of 160 characters. The supported characters are dependent on your carrier, but are character only (not binary). They are not guaranteed to be delivered, may be delivered more than once, and are not guaranteed to be correct (they are subject to corruption).
reboot	Reboot the device immediately. No parameters.
request_ connect	Request a Remote Manager data connection. If a connection has already been made, forces a reconnect of the management session. No parameters.
command	Send a command. Options include: **maxResponseSize=''''** Set maximum size of the response. If the response is larger than the **maxResponseSize**, the response is truncated. The value is the number of SMS messages that make up the response. This is an optional parameter. If not specified, the size of the response is determined by the device.
user_msg	Send a message to a registered listener in python. This command is similar to the RCI do_command custom target command. **path=''''** Send requests to a specific listener. If this optional parameter is omitted, the message is delivered to the listener on the device that is registered to the null path.
dia	Send a DIA command to the running DIA instance. The format of this request is documented in the DIA SMS presentation documentation.

Command	Description
Optional attribute for the above commands	**format="base64|text"** Set the encoding of the request to Base64 or text: use **base64** for Base64 encoding; use **text** for plain text. The default format is **text**. All subcommands (cli, user_msg, and so on) support the format="base64|text" attribute. If the server detects characters that will cause the response to be invalid XML, it encodes the response in Base64 and indicates this with the format="base64" attribute. That is, even if a request uses format="text", the reply may have format="base64" set.

Using SMS to send a request connect

You can use Remote Manager to send an SMS **request connect** message to a Remote Manager-registered device to cause the device to connect back to the server using the device-initiated connection method over TCP/IP. Once the device is connected, you can initiate web services requests and Remote Manager UI actions for the device. In this way, devices do not need to maintain a Remote Manager connection at all times. Instead, connections can be established dynamically as needed.

Provision device for SMS

To use SMS with a device, Remote Manager needs the telephone number of the device. Typically, when a registered device connects for the first time, the telephone number is read from the device and automatically provisioned. However, there are cases where auto-provisioning does not occur. For example, a device connects for the first time and cellular is not yet configured or a device is provisioned before connecting to Remote Manager. In these cases, you must manually provision the device with the telephone number.

To provision the telephone number for a Remote Manager-registered device, the telephone number must be added to an entry in the NetworkInterface that represents a SIM installed on the device. To provision a device, follow these general steps:

Step1: Retrieve the telephone number from a device

Step 2: Find the NetworkInterface record for the device

Step 3: Update NetworkInterface record

Wait for Device to Connect

Step1: Retrieve the telephone number from a device

You can retrieve the telephone number of the device using the following RCI request. The sample response contains the telephone number of the device (phnum):

```
<rci_request version="1.1">
    <query_state>
       <mobile_stats />
    </query_state>
</rci_request>
<rci_reply version="1.1">
    <query_state>
       <mobile_stats>
          <mobile_version>1.1</mobile_version>
          <modemtype>GSM</modemtype>
          <rssi>-42</rssi>
          <quality max="5">5</quality>
```

```
<g3rssi>0</g3rssi>
<g3quality max="5">0</g3quality>
<rstat code="1">Registered (Home Network)</rstat>
<cid>34016</cid>
<lac>32004</lac>
<imsi>310410316937398</imsi>
<iccid>89014104243169373988</iccid>
<phnum>19522213895</phnum>    <!-- phone number of the device -->
<manuf>SIEMENS</manuf>
<model>TC63</model>
<sn>355633002498656</sn>
<rev>REVISION 02.000</rev>
<varinfo index="1">
   <desc>Network Name</desc>
   <data>N/A</data>
</varinfo>
<varinfo index="2">
   <desc>(E)GPRS Status</desc>
   <data>GPRS Attached</data>
</varinfo>
<varinfo index="3">
   <desc>Current Band</desc>
   <data>850 MHz, 1900 MHz</data>
</varinfo>
<varinfo index="4">
   <desc>User Band Selection</desc>
   <data>Automatic</data>
</varinfo>
<varinfo index="5">
   <desc>Mobile Channel</desc>
   <data>235</data>
</varinfo>
<varinfo index="6">
   <desc>Mobile Country Code</desc>
   <data>310</data>
</varinfo>
<varinfo index="7">
   <desc>Mobile Network Code</desc>
   <data>410</data>
</varinfo>
<varinfo index="8">
   <desc>User Carrier Selection</desc>
   <data>Automatic</data>
</varinfo>
<varinfo index="9">
   <desc>PLMN Color</desc>
   <data>3</data>
</varinfo>
<varinfo index="10">
   <desc>Base Station Color</desc>
   96
   <data>5</data>
</varinfo>
<varinfo index="11">
   <desc>Max Power RACH</desc>
   <data>0</data>
</varinfo>
<varinfo index="12">
   <desc>Min Rx Level</desc>
```

```
            <data>-111</data>
         </varinfo>
         <varinfo index="13">
            <desc>Base Coefficient</desc>
            <data>68</data>
         </varinfo>
         <varinfo index="14">
            <desc>SIM Status</desc>
            <data>5: SIM Initialization Complete</data>
         </varinfo>
         <varinfo index="15">
            <desc>SIM PIN Status</desc>
            <data>Ready</data>
         </varinfo>
         <stats_index>5</stats_index>
         <multi_sim_enabled>no</multi_sim_enabled>
      </mobile_stats>
   </query_state>
</rci_reply>
```

Step 2: Find the NetworkInterface record for the device

Note The information within this step only applies to configurations that have an existing entry in NetworkInterface record to update. If you perform the GET below and determine that your configuration does not have a **niId** value, skip this step and proceed to step 4.

To find the NetworkInterface record to update for your Remote Manager-registered device, perform a GET similar to the following:

```
GET /ws/DeviceInterface/?condition=devConnectwareId='00000000-00000000-00409DFF-
FF2EB94D'
```

Replace '00000000-00000000-00409DFF-FF2EB94D' with the device ID of your device.

Here is a sample reply:

```
<result>
   <resultTotalRows>3</resultTotalRows>
   <requestedStartRow>0</requestedStartRow>
   <resultSize>1</resultSize>
   <requestedSize>1000</requestedSize>
   <remainingSize>0</remainingSize>
   <DeviceInterface>
      <id>
         <devId>6</devId>
         <devVersion>0</devVersion>
         <niId>26</niId>
         <niVersion>0</niVersion>
      </id>
      <devRecordStartDate>2011-01-13T18:22:00Z</devRecordStartDate>
      <devMac>00:40:9D:2E:B9:4D</devMac>
      <devCellularModemId>355633002498656</devCellularModemId>
      <devConnectwareId>00000000-00000000-00409DFF-FF2EB94D</devConnectwareId>
      <cstId>10</cstId>
      <grpId>10</grpId>
      <devEffectiveStartDate>2011-01-05T21:37:00Z</devEffectiveStartDate>
      <devTerminated>false</devTerminated>
      <niRecordStartDate>2011-02-15T21:45:00Z</niRecordStartDate>
```

```
      <niInterfaceType>0</niInterfaceType>
      <niEffectiveStartDate>2011-02-15T20:25:00Z</niEffectiveStartDate>
      <niTerminated>false</niTerminated>
      <niPhone>N/A</niPhone>
      <niActivePhone>false</niActivePhone>
      <niIdigiPhone>32075</niIdigiPhone>
   </DeviceInterface>
</result>
```

Within the result, find the **nild** of the NetworkInterface record to be updated. In the above example, the **nild** is 26. Use the nild to retrieve the NetworkInterface record:

```
/ws/NetworkInterface/26/0 (replace with your device's niId, 0 means most recent
version)
```

```
<result>
   <resultTotalRows>1</resultTotalRows>
   <requestedStartRow>0</requestedStartRow>
   <resultSize>1</resultSize>
   <requestedSize>1000</requestedSize>
   <remainingSize>0</remainingSize>
   <NetworkInterface>
      <id>
         <niId>26</niId>
         <niVersion>0</niVersion>
      </id>
      <niRecordStartDate>2011-02-15T21:45:00Z</niRecordStartDate>
      <devId>6</devId>
      <devVersion>0</devVersion>
      <niInterfaceType>0</niInterfaceType>
      <cstId>10</cstId>
      <grpId>10</grpId>
      <niEffectiveStartDate>2011-02-15T20:25:00Z</niEffectiveStartDate>
      <niTerminated>false</niTerminated>
      <niPhone>N/A</niPhone>
      <niPhoneCarrier>3</niPhoneCarrier>
      <niActivePhone>false</niActivePhone>
      <niIdigiPhone>32075</niIdigiPhone>
      <niIdigiServiceId>idgv</niIdigiServiceId>
   </NetworkInterface>
</result>
```

Step 3: Update NetworkInterface record

To update the NetworkInterface record with the device Modem ID, copy the contents of <NetworkInterface> from the GET above. Update the **niPhone** tag with the phone number you discovered in Step1: Retrieve the telephone number from a device (replace N/A with your device phone number). Change the status of **niActivePhone** and then remove the id tag.

The values added below are:
niActivePhone: true (to indicate this is the active Remote Manager SMS record. There can be more than one NetworkInterface record per device. Only one can have **niActivePhone** true).
niPhone: The phone number of the SIM (discovered in step 1).

```
PUT /ws/NetworkInterface/26
```

```
<?xml version="1.0" encoding="UTF-8"?>
<NetworkInterface>
```

```
    <niRecordStartDate>2011-02-15T21:45:00Z</niRecordStartDate>
    <devId>6</devId>
    <devVersion>0</devVersion>
    <niInterfaceType>0</niInterfaceType>
    <cstId>10</cstId>
    <grpId>10</grpId>
    <niEffectiveStartDate>2011-02-15T20:25:00Z</niEffectiveStartDate>
    <niTerminated>false</niTerminated>
    <niPhone>19522213895</niPhone>
    <niActivePhone>true</niActivePhone>
</NetworkInterface>
```

Step 4: Configure phone number without an existing NetworkInterface record

Note The information within this step only applies to configurations that do not have an existing entry in NetworkInterface to update (as described in step 2).

Find the the devId for your Remote Manager-registered device (indicated in the example below):

Perform a GET on /ws/DeviceCore?condition=devConnectwareId='00000000-00000000-00409DFF-FF4A3946' (replace with your device ID).

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
   <resultTotalRows>1</resultTotalRows>
   <requestedStartRow>0</requestedStartRow>
   <resultSize>1</resultSize>
   <requestedSize>1000</requestedSize>
   <remainingSize>0</remainingSize>
   <DeviceCore>
      <id>
         <devId>1224</devId>  <!-- devId of the device -->
         <devVersion>28</devVersion>
      </id>
      <devRecordStartDate>2011-12-20T20:34:00.000Z</devRecordStartDate>
      <devMac>00:40:9D:4A:39:46</devMac>
      <devCellularModemId>357975020409993</devCellularModemId>
      <devConnectwareId>00000000-00000000-00409DFF-FF4A3946</devConnectwareId>
      <cstId>27</cstId>
      <grpId>27</grpId>
      <devEffectiveStartDate>2011-12-20T20:23:00.000Z</devEffectiveStartDate>
      <devTerminated>false</devTerminated>
      <dvVendorId>4261412864</dvVendorId>
      <dpDeviceType>ConnectPort X4</dpDeviceType>
      <dpFirmwareLevel>34406404</dpFirmwareLevel>
      <dpFirmwareLevelDesc>2.13.0.4</dpFirmwareLevelDesc>
      <dpRestrictedStatus>0</dpRestrictedStatus>
      <dpLastKnownIp>10.8.16.16</dpLastKnownIp>
      <dpGlobalIp>66.77.174.126</dpGlobalIp>
      <dpConnectionStatus>1</dpConnectionStatus>
      <dpLastConnectTime>2011-12-20T20:24:00.000Z</dpLastConnectTime>
      <dpContact />
      <dpDescription />
      <dpLocation />
      <dpPanId>0xd367</dpPanId>
      <xpExtAddr>00:13:A2:00:40:66:A1:B2</xpExtAddr>
      <dpServerId>ClientID[3]</dpServerId>
      <dpZigbeeCapabilities>383</dpZigbeeCapabilities>
```

```
   </DeviceCore>
</result>
```

Update the devId, niInterfaceType, and niPhone tags

- Update the devId tag below with the devId number discovered in step a, devId is 1224 in the above example. Replace the devId number in the example below with your device's devId number.
- Ensure the the niInterfaceType value is set to 0.
- Update the niPhone tag below with the phone number (found within the phnum tag) discovered in step 1. Replace the phone number displayed in the example below with your device's phone number.

The values added below are:
devId: The devID number of the device.
niPhone: The phone number of the device (discovered in step 1).
niInterfaceType: The interface type of the device (0 means most recent version).

POST /ws/NetworkInterface

```
<NetworkInterface>
   <devId>1224</devId>
   <niInterfaceType>0</niInterfaceType>
   <niTerminated>false</niTerminated>
   <niPhone>19522213895</niPhone>
</NetworkInterface>
```

Configure Device to Receive SMS Commands

The following example RCI request will configure a device to enable SMS, configure SMS, disable client initiated Remote Manager connections, and configure paged Remote Manager connections. See below for an explanation of the parameters.

RCI request:

```
<rci_request version="1.1">
   <set_setting>
      <smscell>
         <state>on</state>
      </smscell>
      <idigisms>
         <state>on</state>
         <restrict_sender>on</restrict_sender>
         <phnum>32075</phnum>
         <service_identifier>idgt</service_identifier>
      </idigisms>
      <mgmtconnection index="1">
         <connectionType>client</connectionType>
         <connectionEnabled>off</connectionEnabled>
      </mgmtconnection>
      <mgmtconnection index="4">
         <connectionType>paged</connectionType>
         <connectionEnabled>on</connectionEnabled>
         <pagedConnectionOverrideEnabled>on</pagedConnectionOverrideEnabled>
         <serverArray index="1">
            <serverAddress>en://remotemanager.digi.com</serverAddress>
         </serverArray>
      </mgmtconnection>
```

```
      <mgmtglobal>
         <connIdleTimeout>2220</connIdleTimeout>
      </mgmtglobal>
      <mgmtnetwork index="1">
         <networkType>modemPPP</networkType>
         <connectMethod>mt</connectMethod>
         <mtRxKeepAlive>3000</mtRxKeepAlive>
         <mtTxKeepAlive>3000</mtTxKeepAlive>
         <mtWaitCount>3</mtWaitCount>
      </mgmtnetwork>
   </set_setting>
</rci_request>
```

RCI for SMS

RCI group: **idigisms**

Field	Options	Description
state	on, off	Remote Manager SMS support enabled or disabled. If off, SMS messages will not be processed.
phnum	number	The phone number that the device will use to send messages to Remote Manager. This needs to be obtained from Digi (each cluster has its own phone number).
service_ identifier	string	An ID that when combined with the phone number forms the complete address of Remote Manager server. This needs to be obtained from Digi (each cluster has its own phone number).
restrict_ sender	on, off	If on, only Remote Manager SMS messages originating from "phnum" and with the service ID "service_identifier" will be honored as Remote Manager SMS messages.

RCI group: **smscell**

Field	Options	Description
state	on, off	Enables basic SMS support for a device. This needs to be on for Remote Manager SMS to communicate.

RCI group: **mgmtconnection index = "1"** (client initiated Remote Manager connections)

Field	Options	Description
connectionEnabled	on, off	Enables client initiated connections. When off, the device will not attempt to keep a Remote Manager connection open.

RCI group: **mgmtconnection index = "4"** (paged - i.e. temporary - - Remote Manager connections)

Field	Options	Description
connectionEnabled	on, off	Enables temporary connections. A connection request results in a paged Remote Manager connection being established to the server. If this parameter is off, a connection will not be made.
pagedConnectionOverrideEnabled	on, off	When on, paged connections will take priority over client initiated requests so that connection requests always result in a new connection. Set to on.
serverArrayindex="1" serverAddress	url	Send to the dns name of Remote Manager server in the form: `en://<dns-name>`.

RCI group: **mgmtglobal**

Field	Options	Description
connIdleTimeout	Timeout in seconds	Connection is dropped after this number of seconds of inactivity. Any traffic on the connection, including keep-alive traffic, count as non-idle for purposes of this timer.

RCI group: **mgmtnetwork index = "1 "** (cellular Remote Manager connection configuration)

Field	Options	Description
mtRxKeepAlive	Timeout in seconds	Receive keep-alive timeout. Must be higher than connIdleTimeout or connIdleTimeout is defeated.
mtTxKeepAlive	Timeout in seconds	Transmit keep-alive timeout. Must be higher than connIdleTimeout or connIdleTimeout is defeated.
mtWaitCount	Number	Number of missed keep-alives before connection is considered dropped. Shown for completeness only; is not directly related to connection request behavior.

Send Remote Manager SMS Request Connect

To send a connect request to a device via SMS, POST the following SCI request to /ws/sci:

```
<sci_request version="1.0">
   <send_message synchronous="true" syncTimeout="60" reply="all">
      <targets>
         <device id="00000000-00000000-00000000-00000000" />
      </targets>
      <sms>
         <request_connect />
      </sms>
   </send_message>
</sci_request>
```

Details:
SCI is used to send SMS requests to Remote Manager-registered devices. The behavior is very similar to RCI processing from a user's perspective.

As in RCI, web services requests result in jobs being created in Remote Manager. These jobs can be synchronous or asynchronous and job results are retrieved the same way they are for RCI jobs.

The <send_message> command will be used, <send_message> options have the following effect with SMS:

- synchronous= "true|false"

"true " results in a synchronous request; "false" for asynchronous.

- syncTimeout= "x"

Time in seconds that the operation is given to complete. Valid for synchronous jobs only.

- reply= "all|errors|none"

"all " means return a reply SMS,
"errors " means request a reply but the result of the command will only show errors,
"none " means do not request a response.

This controls whether an SMS reply is sent by the device back to the server for a command. This is primarily intended to allow the SMS user to directly control the number of Remote Manager SMS messages being sent, since they are charged for each one. Note, this option is honored even when it results in less than ideal behavior. For instance, a no-reply ping is useless.

SMS requests are specified by the tag `<sms>` as a child element of `<send_message>`.

<request_connect>, requests a device to connect using EDP (reconnects if already connected).

- request_connect takes no parameters

Wait for Device to Connect

The connection status of any Remote Manager-registered device may be found by performing a GET on /ws/DeviceCore.

The result has an entry for each Remote Manager-registered device. In that entry, the element dpConnectionStatus is 0 if the device is disconnected and 1 if connected:

```
<dpConnectionStatus>0</dpConnectionStatus>
```

Note A GET on /ws/DeviceCore returns a list of all Remote Manager-registered devices. To retrieve status for a single device, issue a GET on /ws/DeviceCore/{id} where the id is the id associated with a particular device.

Send a Disconnect

Once work is complete to a device, a web services client may optionally disconnect the registered device from Remote Manager:

POST /ws/sci

```
<sci_request version="1.0">
   <disconnect>
      <targets>
         <device id="00000000-00000000-00000000-00000000" />
```

```
      </targets>
   </disconnect>
</sci_request>
```

Satellite requests

Remote Manager sends Iridium requests to Remote Manager-registered devices via SCI. Iridium satellite messages are handled in a similar manner to RCI messages. They are specified as a child of the <send_message> command. As in RCI, web services requests cause Remote Manager to create jobs. These jobs can be synchronous or asynchronous and job results are retrieved the same way they are for RCI jobs.

Note The asynchronous option should be used most frequently (<send_message synchronous=false">) because the time required to send and receive Iridium satellite messages is very long compared to other communication mechanisms. It can take as little as minutes and as much as hours to send and receive Iridium satellite messages.

<send_message> options have the following effect:

- **synchTimeout="xxx"**
  Behavior is identical to exsiting SCI requests.

- **reply="all|errors|none"**
  Controls whether a reply is sent by the device back to the server for a command. This is primarily intended to allow you to control the number of Iridium satellite messages being sent directly. The default is "none". Note that "all" and "errors" continue to work as they do currently in other SCI requests.

- **cache="true|false|only"**
  Remote Manager does not currently service Iridium requests from the cache. Therefore, specifying "only" returns an error. In addition, "true" or "false" result in the request being sent to the device. The default is "false".

Iridium requests are specified by the <iridium> tag as a child element of <send_message>.

Iridium satellite command children

Command	Description
ping	Request to determine whether device is able to receive Iridium satellite messages - No parameters

Command	Description
raw	Send a raw Iridium satellite message from a device with no modification from Remote Manager; this method is referred to as "raw". Raw messaging is useful in cases when customers wish to use every byte of the Iridium satellite message (Remote Manager protocol takes approximately 5 bytes per message of overhead), or when using a device that doesn't have Remote Manager protocol support but does have Iridium Satellite support. Raw messages can be no longer than 270 bytes for Iridium satellite messages sent from the server to the device, and no longer than 340 bytes for messages sent from the device to the server.
reboot	Reboot device immediately - No parameters
request_ connect	Request a Remote Manager data connection. If a connection has already been made, this will force a reconnect of the management session. - No parameters
command	Command line interface request **maxResponseSize=''''** Set maximum size of the response. To reduce incurred costs, this value should be set to "1" as often as possible. If the response is larger than this value, it will be truncated. The value is the number of 270 byte length Iridium satellite messages into which the response is broken. This is an optional parameter. If not specified, the size of the response is determined by the device.
user_msg	Send a message to a registered listener in python. This command is similar to the RCI do_command custom target command. **path=''''** Send requests to a specific listener. If this optional command is omitted, the message is delivered to the listener on the device that is registered to the null path.
Optional attribute for the above commands	**format="base64|text"** Set the of encoding base for the request to Base64 or text. "base64" indicates Base64 encoding, and "text" means the request is in plain text. The default for format is text. All subcommands (cli, user_msg, etc.) support the format="base64|text" attribute. **Note** If the server detects characters that will cause the response to be invalid XML, it will encode the response in Base64 and indicate this with the format="base64" attribute. That is, even if a request uses format="text", the reply may have format="base64" set.

Request connect Iridium support

Remote Manager can be used to send an Iridium "request connect" satellite message to a device, which will cause it to connect back to the server using the device-initiated connection method over EDP. This will only be possible if the device has an active TCP/IP connection to the internet, such as a cellular data connection, WiFi or Ethernet connection. Once it is connected, web services requests and UI actions can be made to the device. With this support, devices no longer need to maintain a Remote Manager connection at all times. Connections instead can be established dynamically.

This section describes the web services actions to accomplish this. These actions can be performed within the Remote Manager UI as well.

Configure Remote Manager with the Modem ID of the Device

Remote Manager needs to be informed of the device Modem ID. When a device connects for the first time, the Modem ID is read from it and automatically provisioned. There are cases where auto provisioning will not work (satellite is not configured yet, the device is provisioned without it ever being connected, and so on). A manual provisioning method solves this problem.

To provision the Modem ID of a device in Remote Manager, the Modem ID for the satellite modem is added to a record in the NetworkInterface.

Note The Iridium modem must be powered on; otherwise the Modem ID is not returned in the RCI reply.

Step 1: Retrieve modem ID from device
Step 2: Locate NetworkInterface record
Step 3: Update NetworkInterface record
Step 4: Configure modem ID without a NetworkInterface record

Step 1: Retrieve modem ID from device

The Modem ID of the device can be retrieved using the following RCI request (replace with the device ID for your device):

```
<sci_request version="1.0">
   <send_message>
      <targets>
         <device id="00000000-00000000-0004F3FF-FF03A80C" />
      </targets>
      <rci_request version="1.1">
         <query_state />
      </rci_request>
   </send_message>
</sci_request>
```

Example reply:

Note Within the following reply, the Modem ID number (indicated below) is listed within the <serial_number> tag. Make note of the number. Later in this example you will need to use this number, based on the device ID, to update your configuration. Copy only the Modem ID number (that is, exclude the <serial_number> tags).

```
<?xml version="1.0" encoding="UTF-8"?>
<sci_reply version="1.0">
    <send_message>
        <device id="0000000000000-00000000-0004F3FF-FF03A80C">
```

```
            <rci_reply version="1.1">
                <query_state>
                    <iridium_info>
                        <power>on</power>
                        <serial_number>300234010152270</serial_number>  <!--
modem ID number of the device -->
                        <manufacturer>Iridium</manufacturer>
                        <model>IRIDIUM 9600 Family SBD Transceiver</model>
                        <software_revision>TA10003</software_revision>
                        <signal_strength>3</signal_strength>
                        <network_available>yes</network_available>
                        <rx_msgs>2</rx_msgs>
                        <rx_msg_bytes>10</rx_msg_bytes>
                        <rx_total_bytes>1360</rx_total_bytes>
                        <rx_msg_drops>0</rx_msg_drops>
                        <rx_ring_cnt>2</rx_ring_cnt>
                        <tx_msgs>2</tx_msgs>
                        <tx_msg_bytes>5</tx_msg_bytes>
                        <tx_total_bytes>220</tx_total_bytes>
                    </iridium_info>
                </query_state>
            </rci_reply>
        </device>
    </send_message>
</sci_reply>
```

Step 2: Locate NetworkInterface record

Note The information within this step only applies to configurations that have an existing entry in NetworkInterface to update. If you perform the GET below, and determine that your configuration does not have an niId value, skip this step and proceed to step 4.

To find the NetworkInterface record to update for your Remote Manager-registered device:

Perform a GET on /ws/DeviceInterface/?condition=devConnectwareId='00000000-00000000-0004F3FF-FF03A8A5' (replace with the device ID for your device).

An example reply:

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
   <resultTotalRows>1</resultTotalRows>
   <requestedStartRow>0</requestedStartRow>
   <resultSize>1</resultSize>
   <requestedSize>1000</requestedSize>
   <remainingSize>0</remainingSize>
   <DeviceInterface>
      <id>
         <devId>32993</devId>
         <devVersion>0</devVersion>
         <niId>133</niId>
         <niVersion>0</niVersion>
      </id>
      <devRecordStartDate>2011-12-16T16:31:00.000Z</devRecordStartDate>
      <devMac>00:04:F3:03:A8:A5</devMac>
      <devCellularModemId>356021015870112</devCellularModemId>
      <devConnectwareId>00000000-00000000-0004F3FF-FF03A8A5</devConnectwareId>
      <cstId>2</cstId>
      <grpId>2</grpId>
```

```
      <devEffectiveStartDate>2011-12-14T23:27:00.000Z</devEffectiveStartDate>
      <devTerminated>false</devTerminated>
      <niRecordStartDate>2011-12-15T00:05:00.000Z</niRecordStartDate>
      <niInterfaceType>4</niInterfaceType>
      <niModemId>N/A</niModemId>
      <niEffectiveStartDate>2011-12-15T00:05:00.000Z</niEffectiveStartDate>
      <niTerminated>false</niTerminated>
      <niActivePhone>false</niActivePhone>
   </DeviceInterface>
</result>
```

Find the niId of the NetworkInterface record to be updated; niId is 133 in the above example. Retrieve the NetworkInterface record using the niId number found above:

/ws/NetworkInterface/133/0 (replace with your device niId, 0 means most recent version)

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
   <resultTotalRows>1</resultTotalRows>
   <requestedStartRow>0</requestedStartRow>
   <resultSize>1</resultSize>
   <requestedSize>1000</requestedSize>
   <remainingSize>0</remainingSize>
   <NetworkInterface>
      <id>
         <niId>133</niId>
         <niVersion>1</niVersion>
      </id>
      <niRecordStartDate>2011-12-15T00:05:00.000Z</niRecordStartDate>
      <devId>32993</devId>
      <devVersion>0</devVersion>
      <niInterfaceType>4</niInterfaceType>
      <niModemId>N/A</niModemId>
      <cstId>2</cstId>
      <grpId>2</grpId>
      <niEffectiveStartDate>2011-12-15T00:05:00.000Z</niEffectiveStartDate>
      <niTerminated>false</niTerminated>
      <niActivePhone>false</niActivePhone>
   </NetworkInterface>
</result>
```

Step 3: Update NetworkInterface record

To update the NetworkInterface record with the device Modem ID, copy the contents of <NetworkInterface> from the GET above. Update the niModemId tag with the Modem ID number (found within the serial_number tag) discovered in step 1 (replace N/A with your device Modem ID number). Lastly, remove the <id> tag and all of its sub-tags.

The values added below are:

niModemId: The Modem ID number of the device (discovered in step 1).

PUT /ws/NetworkInterface/133

```
<?xml version="1.0" encoding="UTF-8"?>
<NetworkInterface>
   <niRecordStartDate>2011-12-15T00:05:00.000Z</niRecordStartDate>
   <devId>32993</devId>
   <devVersion>0</devVersion>
   <niInterfaceType>4</niInterfaceType>
   <niModemId>300234010152270</niModemId>
```

```
    <cstId>2</cstId>
    <grpId>2</grpId>
    <niEffectiveStartDate>2011-12-15T00:05:00.000Z</niEffectiveStartDate>
    <niTerminated>false</niTerminated>
    <niActivePhone>false</niActivePhone>
</NetworkInterface>
```

Step 4: Configure modem ID without a NetworkInterface record

Note The information within this step only applies to configurations that do not have an existing entry in NetworkInterface to update (as described in step 2).

Find the the devId for your Remote Manager-registered device (indicated in the example below):

Perform a GET on /ws/DeviceCore?condition=devConnectwareId='00000000-00000000-0004F3FF-FF03A80C' (replace with your device ID).

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
   <resultTotalRows>1</resultTotalRows>
   <requestedStartRow>0</requestedStartRow>
   <resultSize>1</resultSize>
   <requestedSize>1000</requestedSize>
   <remainingSize>0</remainingSize>
   <DeviceCore>
      <id>
         <devId>32847</devId>  <!-- devId of the device -->
         <devVersion>22</devVersion>
      </id>
      <devRecordStartDate>2011-12-20T20:51:00.000Z</devRecordStartDate>
      <devMac>00:04:F3:03:A8:0C</devMac>
      <devCellularModemId>356021015867894</devCellularModemId>
      <devConnectwareId>00000000-00000000-0004F3FF-FF03A80C</devConnectwareId>
      <cstId>2</cstId>
      <grpId>2</grpId>
      <devEffectiveStartDate>2011-12-16T20:37:00.000Z</devEffectiveStartDate>
      <devTerminated>false</devTerminated>
      <dvVendorId>4261412864</dvVendorId>
      <dpDeviceType>ConnectPort X5 R</dpDeviceType>
      <dpFirmwareLevel>34472963</dpFirmwareLevel>
      <dpFirmwareLevelDesc>2.14.2.3</dpFirmwareLevelDesc>
      <dpRestrictedStatus>0</dpRestrictedStatus>
      <dpLastKnownIp>10.9.16.35</dpLastKnownIp>
      <dpGlobalIp>66.77.174.126</dpGlobalIp>
      <dpConnectionStatus>1</dpConnectionStatus>
      <dpLastConnectTime>2011-12-21T12:38:00.000Z</dpLastConnectTime>
      <dpContact />
      <dpDescription />
      <dpLocation />
      <dpMapLat>44.898533</dpMapLat>
      <dpMapLong>-93.416252</dpMapLong>
      <dpServerId>ClientID[11]</dpServerId>
      <dpZigbeeCapabilities>0</dpZigbeeCapabilities>
      <dpTags>,techpubs,</dpTags>
   </DeviceCore>
</result>
```

Update devId and niModemId tags

- Update the devId tag below with the devId number discovered in step a, devId is 32847 in the above example. Replace the devId number in the example below with your device's devId number.
- Update the niModemId tag below with the Modem ID number (found within the serial_number tag) discovered in step 1. Replace the Modem ID number displayed in the example below with your device's Modem ID number.

The values added below are:

devId: The devID number of the device.
niModemId: The Modem ID number of the device (discovered in step 1).

POST /ws/NetworkInterface

```
<NetworkInterface>
   <devId>32847</devId>
   <niInterfaceType>4</niInterfaceType>
   <niTerminated>false</niTerminated>
   <niModemId>300234010152270</niModemId>
</NetworkInterface>
```

Configure Device to Receive Iridium Satelllite Commands

The following example RCI request will configure a Remote Manager-registered device to enable Iridium Satellite Support, configure Iridium Satellite Support, disable client initiated Remote Manager connections, and configure paged Remote Manager connections. See below for an explanation of the parameters.

RCI request:

```
<rci_request version="1.1">
   <set_setting>
      <idigiiridium>
         <state>on</state>
         <poll>0</poll>
         <power_mgmt>off</power_mgmt>
         <power_state>on</power_state>
      </idigiiridium>
      <mgmtconnection index="1">
         <connectionType>client</connectionType>
         <connectionEnabled>off</connectionEnabled>
      </mgmtconnection>
      <mgmtconnection index="4">
         <connectionType>paged</connectionType>
         <connectionEnabled>on</connectionEnabled>
         <pagedConnectionOverrideEnabled>on</pagedConnectionOverrideEnabled>
         <serverArray index="1">
            <serverAddress>en://remotemanager.digi.com</serverAddress>
         </serverArray>
      </mgmtconnection>
      <mgmtglobal>
         <connIdleTimeout>2220</connIdleTimeout>
      </mgmtglobal>
      <mgmtnetwork index="1">
         <networkType>modemPPP</networkType>
         <connectMethod>mt</connectMethod>
         <mtRxKeepAlive>3000</mtRxKeepAlive>
         <mtTxKeepAlive>3000</mtTxKeepAlive>
```

```
         <mtWaitCount>3</mtWaitCount>
      </mgmtnetwork>
   </set_setting>
</rci_request>
```

Note Your <idigiiridium> settings may vary from what is displayed above.

In the above example the device has been configured to have its modem always powered up (power_mgmt=off, power_state=on) and polling disabled (poll=0); however, you may have configured your device to have polling enabled (in case a ring alert is missed). Additionally, if you have configured your device to have power_mgmt=on, your modem will be off and ring alerts will not be visible. The only way the connect request will be received is by a poll (please remember that each poll results in an Iridium data charge).

Send an Iridium Satellite Request Connect

To send a connect request to a Remote Manager-registered device via Iridium, POST the following SCI request to /ws/sci:

```
<sci_request version="1.0">
   <!-- It is suggested Iridium requests be done asynchronously as they can take
a while and requests done synchronously may time out before the response has been
received -->
   <send_message synchronous="false">
      <targets>
         <device id="00000000-00000000-0004F3FF-FF03A8A5" />
      </targets>
      <iridium>
         <request_connect />
      </iridium>
   </send_message>
</sci_request>
```

Details:
SCI is used to send Iridium requests to Remote Manager-registered devices. The behavior is very similar to RCI processing from a user's perspective.

As in RCI, web services requests result in jobs being created in Remote Manager. These jobs can be synchronous or asynchronous and job results are retrieved the same way they are for RCI jobs.

Note The asynchronous option should be used most frequently (<send_message synchronous=false">) because the time required to send and receive Iridium satellite messages is very long compared to other communication mechanisms. It can take as little as minutes and as much as hours to send and receive Iridium satellite messages.

The <send_message> command will be used; <send_message> options have the following effect:

- synchronous="true|false"
  "true" results in a synchronous request; "false" for asynchronous.

Iridium requests are specified by the tag as a child element of <send_message>.

<request_connect>, requests a device to connect using EDP (reconnects if already connected).

- request_connect takes no parameters

Wait for Device to Connect

The connection status of any Remote Manager-registered device may be found by performing a GET on /ws/DeviceCore.

The result has an entry for each Remote Manager-registered device. In that entry, the element dpConnectionStatus is 0 if the device is disconnected and 1 if connected:

```
<dpConnectionStatus>0</dpConnectionStatus>
```

Note A GET on /ws/DeviceCore returns a list of all Remote Manager-registered devices. To retrieve status for a single device, issue a GET on /ws/DeviceCore/{id} where the id is the id associated with a particular device.

Send a Disconnect

Once work is complete to a device, a web services client may optionally disconnect the device from Remote Manager:

POST /ws/sci

```
<sci_request version="1.0">
   <disconnect>
      <targets>
         <device id="00000000-00000000-00000000-00000000" />
      </targets>
   </disconnect>
</sci_request>
```

SM/UDP

The SM/UDP (Short Message/User Datagram Protocol) feature allows devices to leverage the very small data footprint of Remote Manager SM protocol (currently used for SMS and Iridium Satellite messaging) over UDP. However, it is important to note that SM/UDP requests vary greatly from SMS/Iridium requests as SM/UDP requests are not immediately sent to a device. Instead, requests of this type are queued as devices may not be publicly addressable. This creates a way for devices to interact with Remote Manager in a way that is efficient from a data perspective and does not require a persistent connection. This feature enables devices with constrained data plans to keep data traffic to an absolute minimum by only occasionally sending data readings to Remote Manager.

Initially, no requests are queued in the server. A device will send a request to the server and the server will process the request, sending a reply to the device only if the device specified that one should be sent in the request. At some point an SM/UDP request may be targeted for the device. This request can be sent via a Web Services request, using the options within the More Menu of the Devices page, or as the result of a scheduled task. When a device sends an SM/UDP request (known as a datagram) to Remote Manager, Remote Manager will process the request and queue it for delivery to the device. The next time the device sends a message (regardless of whether a reply was specified), Remote Manager will check for queued messages and send them down to the device. For example, if you send the SM/UDP Reboot request to your device, the device will not reboot immediately. Instead, the SM/UDP Reboot request will be queued for the device. The next time an SM/UDP request is sent to the device, Remote Manager will check for queued messages and send the queued SM/UDP Reboot request to the device instructing it to reboot itself. Once a request is queued for a device, it may remain queued for multiple days. Once the request is actually sent to a device, it typically has a timeout of 60 seconds (plus a small window in some circumstances).

Sending and receiving SM/UDP messages via web services

Remote Manager sends SM/UDP requests to Remote Manager-registered devices via SCI. SM/UDP requests are handled in a similar manner to RCI messages. They are specified as a child of the <send_message> command. As in RCI, web services requests cause Remote Manager to create jobs. These jobs can be synchronous or asynchronous and job results are retrieved the same way they are for RCI jobs.

It is suggested that SM/UDP requests be done asynchronously since requests of this type require the device to send a message before a server message can be sent, and therefore can potentially take a long time to complete. Synchronous requests are acceptable in situations where the device is frequently sending messages to the server.

<send_message> options have the following effect with SM/UDP:

- **synchTimeout="xxx"**
  Behavior is identical to existing SCI requests.
- **reply="all|errors|none"**
  Controls whether a reply is sent by the device back to the server for a command. This is primarily intended to allow you to control the number of SM/UDP messages being sent directly. The default is "none". Note that "all" and "errors" continue to work as they do currently in other SCI requests.
- **cache="true|false|only"**
  Remote Manager does not currently service SM/UDP requests from the cache. Therefore, specifying "only" will return an error. In addition, "true" or "false" will result in the request being sent to the device. The default is "false".

SM/UDP requests are specified by the tag <sm_udp> as a child element of <send_message>.

SM/UDP command children

Command	Description
ping	Request to determine whether device is able to receive SM/UDP requests - No parameters
reboot	Reboot device immediately - No parameters
request_ connect	Request a Remote Manager data connection. If a connection has already been made, this will force a reconnect of the management session. - No parameters

Command	Description
command	Command line interface request **maxResponseSize=""""** Set maximum size of the response. If the response is larger than this value, it will be truncated. The value is the number of SM/UDP messages into which the response is broken. This is an optional parameter. If not specified, the size of the response is determined by the device.
user_msg	Send a message to a registered listener in python. This command is similar to the RCI do_command custom target command. **path=""""** Send requests to a specific listener. If this optional command is omitted, the message is delivered to the listener on the device that is registered to the null path.
Optional attribute for the above commands	**format="base64|text"** Set the encoding of the request to Base64 or text."base64" indicates Base64 encoding, and"text" means the request is in plain text. The default for format is text. All subcommands (cli, user_msg, etc.) support the format="base64|text" attribute. **Note** If the server detects characters that will cause the response to be invalid XML, it will encode the response in Base64 and indicate this with the format="base64" attribute. That is, even if a request uses format="text", the reply may have format="base64" set.

Request connect SM/UDP support

Remote Manager can be used to send a SM/UDP "request connect" message to a Remote Manager-registered device which will cause it to connect back to the server using the device-initiated connection method over TCP/IP. Once it is connected, web services requests and UI actions can be made to the device. With this support, devices no longer need to maintain a Remote Manager connection at all times. Connections instead can be established dynamically.

This section describes the web services actions to accomplish this. All of these actions can be performed in the Remote Manager UI as well.

Configure Device to Receive SM/UDP Commands

The following example will configure a Remote Manager-registered device to enable SM/UDP.

POST to /ws/DeviceCore to provision a device with SM/UDP enabled.

```
<DeviceCore>
  <devMac>00:40:9D:00:00:00</devMac>
  <dpUdpSmEnabled>true</dpUdpSmEnabled>
</DeviceCore>
```

The device will be added to your account and configured to enable SM/UDP.

If you want to disable SM/UDP for a device, use the following example.

PUT to /ws/DeviceCore to update the device and disable SM/UDP.

```
<DeviceCore>
  <devConnectwareId>00000000-00000000-00409DFF-FF000000</devConnectwareId>
  <dpUdpSmEnabled>false</dpUdpSmEnabled>
</DeviceCore>
```

After sending this request the device will no longer be configured for SM/UDP.

Message Compression

```
<dpSmCompressionAvailable>true/false</dpSmCompressionAvailable>
```

This configures the server to allow compression to be used for SM requests. Defaults to false, but can be inferred by a device sending a compressed request.

Pack Command

```
<dpSmPackAvailable>true/false</dpSmPackAvailable>
```

This configures the server to allow pack commands to be sent to the device. The pack command allows multiple SM commands to be merged and sent in a single datagram to reduce data usage and overhead. Defaults to false, but can be inferred by a device sending a pack command.

Battery Operated Mode

```
<dpSmBatteryOperated>true/false</dpSmBatteryOperated>
```

This configures the server to send requests to the device in battery operated mode. Battery operated mode can be used for devices that should only recieve a single reply to a request sent to the server, where it was indicated that the device needed a response. This tells the device that it can immediately shut down its network connection to conserve power. This mode implies that the device also supports the pack command. Unless the device requires this mode, it may add unnecessary limitations. Defaults to false.

Send SM/UDP Request Connect

To send a connect request to a device via SM/UDP, POST the following SCI request to /ws/sci:

```
<sci_request version="1.0">
  <!-- It is suggested SM/UDP requests be done asynchronously as they can
  take a while and requests done synchronously may time out before the
  response has been received.  See the Check Request Status Example
  for information on retrieving status and results. -->
  <send_message synchronous="false">
    <targets>
      <device id="00000000-00000000-00000000-00000000"/>
    </targets>
    <sm_udp>
      <request_connect/>
    </sm_udp>
  </send_message>
</sci_request>
```

Details:

SCI is used to send SM/UDP requests to Remote Manager-registered devices. The behavior is very similar to RCI processing.

As in RCI, web services requests result in jobs being created in Remote Manager. These jobs can be synchronous or asynchronous and job results are retrieved the same way they are for RCI jobs.

Note Digi recommends you execute SM/UDP requests asynchronously. Synchronous requests may time out before the response has been received.

The <send_message> command will be used, <send_message> options have the following effect with SM/UDP:

- synchronous="true|false"

"true" results in a synchronous request; "false" for asynchronous.

SM/UDP requests are specified by the tag <sm_udp> as a child element of <send_message>.

<request_connect>, requests a device to connect using EDP (reconnects if already connected).

- request_connect takes no parameters

Wait for Device to Connect

The connection status of any Remote Manager-registered device may be found by performing a GET on /ws/DeviceCore.

The result has an entry for each Device Cloud-registered device. In that entry, the element dpConnectionStatus is 0 if the device is disconnected and 1 if connected:

```
<dpConnectionStatus>0</dpConnectionStatus>
```

Note: A GET on /ws/DeviceCore returns a list of all Device Cloud-registered devices. To retrieve status for a single device, issue a GET on /ws/DeviceCore/{id} where id is the id associated with a particular device.

Send a Disconnect

Once work is complete to a device, a web services client may optionally disconnect the registered device from Device Cloud:

POST /ws/sci

```
<sci_request version="1.0">
   <disconnect>
      <targets>
         <device id="00000000-00000000-00000000-00000000" />
      </targets>
   </disconnect>
</sci_request>
```

security

You can use the **security** web service to set or remove a password for legacy devices. However, Digi recommends using the v1/devices (for example, **ws/v1/devices/inventory**) APIs to set or change security credentials for one or more devices.

When you set a password for a device, Remote Manager attempts to configure the device with the new password. Until Remote Manager successfully configures the new password, the device is allowed to connect with the previous password. Once Remote Manager has configured the new password on the device, subsequent device connections require the new password.

When you remove a password, Remote Manager removes the password from the Remote Manager server and subsequent device connections do not require a password. However, the device is still configured with the password, but Remote Manager does not require the password for connections.

URI

```
http://<hostname>/ws/security
```

Formats

HTTP method	Format	Description
POST	/ws/security	Set or remove a password for a device.

Elements

Request content for setting a password

```
<set_password>
   <device id="00000000-00000000-00000000-00000000">
      <password>newPassword</password>
   </device>
</set_password>
```

Request content for removing a password

```
<remove_password>
   <device id="00000000-00000000-00000000-00000000" />
</remove_password>
```

You can specify multiple device elements per request.

Task

Use the Task web service to execute a specific task , get information about tasks, or to remove a task. A task is a chain of commands stored in the Remote Manager file system as an XML file. Each command element of a task can specify a command payload, along with events related to its execution.

Once a schedule runs, it creates a task, which is essentially an invocation of a task template. The tasks created by schedules can be managed through the Task API. The tasks, which execute commands, create SCI jobs within the system on a per-command basis. You can check the status of each command by querying the SCI web service interface for a given job.

URI

```
http://ws/Task
```

Formats

HTTP method	Format	Description
GET	/ws/Task	Get a list of all tasks; get a list of tasks for a schedule; get details for a specific task.
GET	/ws/Task/{tskId}	Get details for a specific task.

Elements

tskId

System-generated identifier for the task.

schId

System-generated identifier for the schedule.

tskScheduledTime

Time at which the task is scheduled to run.

tskStartTime

Time at which the task started execution.

tskEndTime

Time at which task stopped executing.

tskTargets

List of targets for the task.

tskSuccess

Number of devices that have successfully completed the task.

tskFailures

Number of devices that have completed the task with an error.

tskStatus

Current status of the task:

0 = new
1 = in_progress
3 = complete
4 = canceled

tskRequestPayload

Request payload of the task.

tskTargetCount

Total number of devices to which the task is targeted.

Example: Get a list of all tasks

The following example shows how to get a list of all tasks for your account.

Request

```
GET /ws/Task
```

Response

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
  <resultTotalRows>1</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>1</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <Task>
    <tskId>732</tskId>
    <schId>258</schId>
    <cstId>2</cstId>
    <usrId>9</usrId>
    <tskScheduledTime>2014-12-14T20:59:58.070Z</tskScheduledTime>
    <tskStartTime>2014-12-14T20:59:58.200Z</tskStartTime>
    <tskTargets>00000000-00000000-886123FF-FF000026</tskTargets>
    <tskSuccess>0</tskSuccess>
    <tskFailures>0</tskFailures>
    <tskStatus>1</tskStatus>
    <tskRequestPayload>
      <description>Another Schedule</description>
      <command>
        <name>List Files</name>
        <event>
          <on_error>
            <end_task/>
          </on_error>
        </event>
        <sci>
          <file_system allowOffline="true">
            <commands>
              <ls path="/"/>
            </commands>
          </file_system>
        </sci>
      </command>
    </tskRequestPayload>
    <tskTargetCount>1</tskTargetCount>
    <tskDescription>Another Schedule</tskDescription>
  </Task>
</result>
```

Example: Get details for a task

The following example shows how to get details for a task with the ID of 732.

Request

```
GET /ws/Task/732
```

Response

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
  <resultTotalRows>1</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>1</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <Task>
    <tskId>732</tskId>
    <schId>258</schId>
    <cstId>2</cstId>
    <usrId>9</usrId>
    <tskScheduledTime>2014-12-14T20:59:58.070Z</tskScheduledTime>
    <tskStartTime>2014-12-14T20:59:58.200Z</tskStartTime>
    <tskTargets>00000000-00000000-886123FF-FF000026</tskTargets>
    <tskSuccess>0</tskSuccess>
    <tskFailures>0</tskFailures>
    <tskStatus>1</tskStatus>
    <tskRequestPayload>
      <description>Another Schedule</description>
      <command>
        <name>List Files</name>
        <event>
          <on_error>
            <end_task/>
          </on_error>
        </event>
        <sci>
          <file_system allowOffline="true">
            <commands>
              <ls path="/"/>
            </commands>
          </file_system>
        </sci>
      </command>
    </tskRequestPayload>
    <tskTargetCount>1</tskTargetCount>
    <tskDescription>Another Schedule</tskDescription>
  </Task>
</result>
```

Example: Upload a task definition

The following example shows how to upload a task definition using the FileData web service.

```
PUT ws/FileData/~/my_tasks/my_task.xml?type=file
```

File my_task.xml contains a valid task definition.

Example: Get a list of jobs for a schedule

The following example shows how to get a list of jobs associated with a specific schedule using the SCI web service.

```
GET ws/sci?condition=schId={schId}
```

Task template

A task consists of one or more commands. The required XML format for a task is as follows:

```
<task>
   <description>My first task.</description>
   <command>
      <name>Reboot</name>
      <event>
         <on_end>
            <sleep value="15" />
         </on_end>
         <on_error>
            <retry count="5" />
         </on_error>
      </event>
      <sci>
         <reboot />
      </sci>
   </command>
   .
   .
   .
   <command>
   </command>
</task>
```

Elements

description

(Optional) Specifies a description for the task.

command

(Required) Provides configuration information for each command in the task. If the task contains multiple commands, specify the command elements in the order in which you want to execute the commands.

name

(Optional) Specifies a name for the command.

event

(Optional) Specifies actions to take for two events: **on_end** and **on_error**.

on_end

Specifies the number of seconds to sleep before starting execution of the next command in the task. The sleep option forces Remote Manager to delay the execution of the next command in the list for the specified number of seconds immediately following the completion of the command. The default is 0 which indicates Remote Manager does not delay after completing command before executing the next command in the task.

Example:

```
<on_end> <sleep value="15"/> </on_end>
```

on_error

Specifies the action to take if an error occurs during command execution: retry, end_task, or continue.

retry: Specifies the number of times to retry command execution. After the specified number of retries, Remote Manager continues with the next command in the task.

end_task: Remote Manager immediately ends execution of the task.

continue: Remote Manager ignores the error and continues with the next command in the task.

The default is **continue**. That is, if you do not specify an **on_error** action, Remote Manager ignores the error and continues with the next command in the task.

Example:

```
<on_error>
   <retry count="5" />
</on_error>
```

v1/alerts

Use the **v1/alerts** web service to create, update, list, and manage alerts for your account. For each alert, specify the alert type, the scope for the alert, as well as rules for the alert to fire and reset. The alert scope and fire/reset rules vary depending on the alert type.

Note In the classic Remote Manager interface and pre-v1 APIs, **alerts** are called **alarms**, and the two terms are synonymous. Going forward, Digi recommends you use the **/v1/alerts** API rather than the pre-v1 Alarm APIs to manage alerts.

URI

```
https://<hostname>/ws/v1/alerts
```

Formats

HTTP method	Format	Description	Parameters
GET	/ws/v1/alerts	Get a summary of the alerts API.	
GET	/ws/v1/alerts/summary	Get a summary of alerts.	query
GET	/ws/v1/alerts/bulk	List alerts in bulk CSV format.	orderby query

HTTP method	Format	Description	Parameters
GET	/ws/v1/alerts/bulk/summary ws/v1/alerts/bulk/status	Export up to 20,000 rows of CSV data representing the summary or status of any alarm that has been fired and/or reset.	orderby query
GET	/ws/v1/alerts/inventory	Example: List alerts	cursor orderby size query
POST	/ws/v1/alerts/inventory	Example: Create one or more alerts	None
GET,PUT,DELETE	/ws/v1/alerts/inventory/{id}	Get a specific alert, modify an existing alert, and delete an alert.	
GET	/ws/v1/alerts/history/{alert_id_source}	Get history of fired alerts.	cursor size start_time end_time order
GET	/ws/v1/alerts/status	Get current status for alerts.	cursor orderby size query
GET	/ws/v1/alerts/summary	Get a summary of all alerts.	cursor orderby size query

Fields

id

System generated identifier for the alert.

name

String name of the alert.

description

String that describes the alert.

enabled

Boolean indicating if the alert is enabled. Default is true.

fire

The definition of the rule describing conditions for when the alert fires. Not all alerts require a fire rule.

fire.parameters

A map of parameter names and values indicating the conditions required for the alert to fire.

priority

String describing the priority assigned to the alert.

- high
- medium
- low

reset

The definition of the rule describing conditions for when the alert is reset. Not all alerts require a reset rule.

reset.enabled

Boolean indicating if the rule should automatically reset.

reset.parameters

A map of parameter names and values indicating the conditions required for the alert to reset.

scope

The definition of the scope defining how the alert is applied.

scope.type

What type of scope defines how the alert is applied.

- **Device**: The target of the alert is a specific device.
- **Global**: The target of the alert is global to the account.
- **Group**: The target of the alert is a group or all devices in a group.
- **Resource**: The target of the alert is a resource in the system.
- **XBeeNode**: The target of the alert is a specific XBee node.

scope.value

The value indicating the scope target depends on the **scope.type** setting.

Scope Type	Scope Value
Device	The scope.value field is a device id. For example: 00000000-00000000-00000000-00000000
Global	The scope.value field is the empty string.
Group	The scope.value field is a group name: For example: Regions/Northwest indicates any devices in the Northwest subgroup of the Regions group.

Scope Type	Scope Value
Resource	The scope.value field is the resource identifier. For example a stream name: 00000000-00000000-00000000-00000000/metrics/sys/cpu/used. Can use the wildcard character (*) to include all path components of a stream name, for example */metrics/sys/cpu/used.
XBeeNode	The scope.value field is an XBee node extended address. For example: 00:08:A2:A5:6E:4D:52:85

type

String that indicates the alert type.

- **DataPoint On Change**: Fires when a data point is uploaded containing a different value from the previous data point. The scope type of a DataPoint on Change alert is always Resource.
- **DataPoint condition**: Fires when a data point is uploaded containing a value that matches the specified criteria. The scope type of a DataPoint condition alert is always Resource.
- **Device Excessive Disconnects**: Fires when devices disconnect with a specified frequency. The scope type of a Device Excessive Disconnects alert is Group or Device.
- **Device Name On Change**: Fires when a devices name changes. The scope type of a Device Name On Change alert is Group or Device.
- **Device Offline**: Fires when devices disconnects for a specified time. The scope type of a Device Offline alert is Group or Device.
- **Dia channel data point condition match**: Fires when a DIA channel data point is uploaded containing a value that matches the specified criteria. The scope type of a Dia channel data point condition match is Group or Device.
- **Missing DataPoint**: Fires when a new data point is not uploaded for a specified time. The scope type of a Missing DataPoint alert is always Resource.
- **Missing DiaChannel DataPoint**: Fires when a new DIA channel data point is not uploaded for a specified time. The scope type of a Missing DiaChannel DataPoint alert is Group or Device.
- **Missing Smart Energy DataPoint**: Fires when a new Smart Energy data point is not uploaded for a specified time. The scope type of a Missing Smart Engergy DataPoint alert is Group, Device, or XBeeNode.
- **Smart energy data point condition match**: Fires when a Smart Energy data point is uploaded containing a value that matches the specified criteria. The scope type of a Smart energy data point condition match ialert s Group, Device, or XBeeNode.
- **Subscription Usage**: Fires when the account subscription usage matches the specified criteria. The scope type of a Subscription Usage alert is always Global.
- **XBeeNode Excessive Deactivations**: Fires when XBee nodes deactivate with a specified frequency. The scope type of an XBeeNode Excessive Deactivations alert is Group, Device ,or XBeeNode.
- **XBeeNode Offline**: Fires when an XBee node is offline for a specified time. The scope type of an XBeeNode Offline alert is Group, Device, or XBeeNode.

Example: Datapoint condition alert

The following example alert payload represents a DataPoint condition alert. This alert might be used to determine if action (replacing a battery) is needed for a device that is tracking voltage.

The alert fires when the value uploaded to the data stream ***/drivestats/voltage** is less than **10** volts for **30** minutes. The alert resets when the voltage value is greater than or equal to **10** volts for **15** minutes (for example, the battery was replaced or charged).

The wildcard (*****) character used in the **scope.value** field indicates that any device uploading this stream is the target for the alert (the **Device ID** is always the first component of a stream name and this alert targets data streams.

```
{
        "description": "Fires when a vehicle fleet battery needs replacing",
        "enabled": true,
        "fire": {
                "parameters": {
                        "thresholdValue": "10",
                        "type": "numeric",
                        "operator": "<",
                        "timeout": "30",
                        "timeUnit": "minutes"
                }
        },
        "id": 150027,
        "name": "Low Voltage",
        "priority": "high",
        "reset": {
                "parameters": {
                        "thresholdValue": "10",
                        "type": "numeric",
                        "operator": ">=",
                        "timeout": "15",
                        "timeUnit": "minutes"
                }
        },
        "scope": {
                "type": "Resource",
                "value": "*/drivestats/voltage"
        },
        "type": "DataPoint condition"
}
```

Example: List alerts

Use the GET **v1/alerts/inventory** web service to retrieve a list of alerts.

URI

```
https://<hostname>/ws/v1/alerts/inventory
```

API Usage

There is no request payload for the GET **ws/v1/alerts/inventory** API. Use the alert fields and queris to select alerts.

Parameters

Name	Type	Description
cursor	string	Cursor to get the next page of alerts. Omit on initial call.
orderby	string	Return alerts ordered by the field given. The default is "name asc"
query	string	Use a query expression to target a specific list of alerts. The default is all alerts.
size	integer	Number of items to return. The maximum and default is 1000.

Successful Response

Upon success, the list of alerts is returned.

```
{
  "count": 1,
  "size": 1000,
  "list": [
    {
      "description": "Detects when the office cellular backup network goes
offline",
      "enabled": true,
      "fire": {
        "parameters": {
          "reconnectWindowDuration": "1"
        }
      },
      "id": 146639,
      "name": "Office Backup Offline",
      "priority": "high",
      "reset": {},
      "scope": {
        "type": "Device",
        "value": "00000000-00000000-00000000-00000000"
      },
      "type": "Device Offline"
    }
  ]
}
```

Error Response

An error response returns the **error_status** and **error_message** fields in the payload.

```
{
  "error_status": 400,
  "error_message": "error message"
}
```

Example: Create one or more alerts

Use the POST **v1/alerts/inventory** web service to create alerts for your account.

URI

```
https://<hostname>/ws/v1/alerts/inventory
```

API Usage

There are no parameters for the POST **ws/v1/alerts/inventory** API.

Specify an HTTP header of **Content-Type: application/json** and post **json** payload describing the alert or alerts.

Request Payload

To create multiple alerts, use a list syntax. For a single alert, the list syntax is not required.

```
[
        { ...alert1 },
        { ...alert2 },
        ...
        { ...alertN }
]
```

Successful Response

Upon success, the alert is returned. The returned alert may differ from the input alert because default values are supplied in the returned alert. A list response is returned whether the POST created single or multiple alerts.

```
{
        "count": N,
        "list" : [
                { ...alert1 },
                { ...alert2 },
                ...
                { ...alertN },
        ]
}
```

Error Response

When creating multiple alerts, if any alert is successfully created, a response of 207 and error information for the failed responses are returned.

In the following example response payload, the second alert failed. The HTTP response is 207 because one alert (alert1) was created successfully, and the second item in the list has error values indicating the problem with the second alert (alert2).

```
{
        "count": N,
        "list": [
                { ...alert1 },
                {
                "error_context": { ...alert2 },
                "error_message": "The error message",
                "error_status": 400
                },
                { ...alertN }
        ]
}
```

Sample Alert Definitions

The following sample payloads represent the different alert types.

DataPoint condition

DataPoint On Change

Device Excessive Disconnects

Device Name On Change

Device Offline

Dia channel data point condition match

Missing DataPointThe following payload creates a Missing DataPoint alert. This alert type fires when the interval between data point uploads or data point measurements exceeds the requested time. Devices may cache data uploads, uploading infrequently (for example 24 hours), but uploading many measurements (for example, 24 total measurements if sampling every hour). This alert automatically resets when data is reported.Values: scope.type: Always Resource.scope.value: The stream with wildcard (*) for device id or path components.uploadInterval: Fire if the amount of time between uploads exceeds this interval.uploadTimeUnit: The units of the uploadInterval value—seconds, minutes, hours.readingInterval: Fire if the amount of time between individual samples/reading exceeds this interval. readingInterval}} value—seconds, minutes, hours.reset.disabled: Resets automatically when data is reported, you can disable this by specifying true.{ "name":"Fleet sensor not reporting", "type":"Missing DataPoint", "scope":{ "type":"Resource", "value":"*/drivestats/voltage" } "fire":{ "parameters":{ "readingTimeUnit":"seconds", "uploadTimeUnit":"minutes", "uploadInterval":"1", "readingInterval":"5" } }, "reset": { "disabled": true } }

Missing DiaChannel DataPoint

Missing Smart Energy DataPoint

Smart energy data point condition match

Subscription Usage

XBeeNode Excessive Deactivations

XBeeNode Offline

DataPoint On Change

The following payload creates a **DataPoint On Change** alert. This alert type fires when a data stream value changes. Repeated uploads of the same value do not cause the alert to fire. This alert automatically resets when the data point value remains unchanged for 60 minutes.

The target data stream matches any device with a **drivestats/voltage** stream name.

Values:

- **scope.type**: Always **Resource**.
- **scope.value**: The stream with wildcard (*) for device id or path components.
- **timeout**: Reset automatically if the value does not change for this amount of time.
- **timeUnit**: One of **seconds**, **minutes**, **hours**.

```
{
	"type":"DataPoint On Change",
	"reset": {
		"parameters": {
			"timeout": "60",
			"timeUnit": "minutes"
		}
	},
	"scope": {
		"type":"Resource",
		"value":"*/drivestats/voltage"
```

```
        }
}
```

DataPoint condition

The following payload creates a **DataPoint condition** alert. This alert type fires when a data stream value matches the condition criteria.

Values:

- **scope.type**: Always **Resource**.
- **scope.value**: The stream with wildcard (*) for device id or path components.
- **timeout**: Fire or reset if the value matches for this amount of time.
- **timeUnit**: One of **seconds**, **minutes**, **hours**.
- **operator**: One of **<**, **<=**, **>**, **>=**, **=**, **!=**.
- **type**: One of **string** or **numeric**.
- **thresholdValue**: The target value for the condition.

```
{
        "type": "DataPoint condition",
        "name": "Voltage check",
        "description": "Watch for low voltage on fleet vehicles",
        "scope": {
                "type": "Resource",
                "value": "*/drivestats/voltage"
        },
        "fire": {
                "parameters": {
                "operator": "<",
                "thresholdValue": "10",
                "timeUnit": "minutes",
                "timeout": "60",
                "type": "numeric"
                }
        },
        "reset": {
                "parameters": {
                "operator": ">=",
                "thresholdValue": "10",
                "timeUnit": "minutes",
                "timeout": "60",
                "type": "numeric"
                }
        }
}
```

Device Excessive Disconnects

The following payload creates a **Device Excessive Disconnects** alert. This alert type fires when a device goes offline many times during a time period.

Values:

- **scope.type**: One of **Group** or **Device**.
- **scope.value**: If type is **Group**, then value is the full group path (with no leading slash). If type is **Device**, then value is the full device ID. If type is **XBeeNode**, then the value is the XBee node extended **src** address.
- **disconnectCount**: Fires if the device disconnects more than this many times in the **disconnectWindow**
- **disconnectWindow**: The period of time in minutes for which to count disconnects.
- **reconnectWindow**: Resets when the device connects and stays connect for this number of minutes.

```
{
        "type":"Device Excessive Disconnects",
        "scope":{
                "type":"Device",
                "value":"00000000-00000000-000000FF-FF000000"
        }
        "fire":{
                "parameters":{
                        "disconnectCount":"10",
                        "disconnectWindow":"60"
                }
        },
        "reset":{
                "parameters":{
                        "reconnectWindow":"30"
                }
        },
}
```

Device Name On Change

The following payload creates a **Device Name on Change** alert. This alert type fires when a device name changes. The alert does not fire when a device name is set on a device that has no name. The alert can reset automatically.

Values:

- **scope.type**: One of **Group** or **Device**.
- **scope.value**: If type is **Group**, then value is the full group path (with no leading slash). If type is **Device**, then value is the full device ID. If type is **XBeeNode**, then the value is the XBee node extended **src** address.
- **timeout**: Reset automatically if the value does not change for this amount of time.
- **timeUnit**: One of **seconds**, **minutes**, **hours**.

```
{
        "type":"Device Name On Change",
        "reset":{
                "parameters":{
                        "timeout":"12",
                        "timeUnit":"hours"
```

```
                }
        },
        "scope":{
                "type":"Group",
                "value":"Deployments/Europe/UK"
        }
}
```

Device Offline

The following payload creates a **Device Offline** alert. This alert type fires when a device disconnects for a period of time.

Values:

- **scope.type**: One of **Group** or **Device**.
- **scope.value**: If type is **Group**, then value is the full group path (with no leading slash). If type is **Device**, then value is the full device ID. If type is **XBeeNode**, then the value is the XBee node extended **src** address.
- **reconnectWindowDuration**: Fires if the device does not reconnect within this number of minutes.
- **reset.disabled**: Is set to true to cause this alert to not auto-reset. The default is false.

```
{
        "name":"Network offline",
        "type":"Device Offline",
        "description":"Detect if any midwest office networks are offline",
        "priority":"high",
        "fire":{
                "parameters":{
                "reconnectWindowDuration":"12"
                }
        },
        "reset": {
                "disabled": true
        },
        "scope": {
                "type":"Group",
                "value":"Stores/North America/Midwest"
        }
}
```

Dia channel data point condition match

The following payload creates a **Dia channel data point condition match** alert. This alert type fires when the value reported in a DIA channel matches the condition criteria.

Values:

- **scope.type**: One of **Group** or **Device**.

- **scope.value**: If type is **Group**, then value is the full group path (with no leading slash). If type is **Device**, then value is the full device ID. If type is **XBeeNode**, then the value is the XBee node extended **src** address.
- **channelName**: The dia channel name or wildcard (*) for all channels.
- **instanceName**: The dia channel instance name or wildcard (*) for all instances.
- **timeout**: Fire or reset if the value matches for this amount of time.
- **timeUnit**: One of **seconds**, **minutes**, **hours**.
- **operator**: One of **<**, **<=**, **>**, **>=**, **=**, **!=**.
- **type**: One of **string** or **numeric**.
- **thresholdValue**: The target value for the condition.

```
{
	"type": "Dia channel data point condition match",
	"scope": {
		"type": "Group",
		"value": "New York/Sensors"
	},
	"fire": {
		"disabled": false,
		"parameters": {
			"channelName": "chan1",
			"instanceName": "stream",
			"operator": ">=",
			"thresholdValue": "25",
			"timeUnit": "seconds",
			"timeout": "30",
			"type": "numeric"
		}
	},
	"reset": {
		"disabled": false,
		"parameters": {
			"channelName": "chan1",
			"instanceName": "stream",
			"operator": "<",
			"thresholdValue": "25",
			"timeUnit": "seconds",
			"timeout": "30",
			"type": "numeric"
		}
	}
}
```

Missing DataPoint

The following payload creates a **Missing DataPoint** alert. This alert type fires when the interval between data point uploads or data point measurements exceeds the requested time. Devices may cache data uploads, uploading infrequently (for example 24 hours), but uploading many measurements (for example, 24 total measurements if sampling every hour).
This alert automatically resets when data is reported.

Values:

- **scope.type**: Always **Resource**.
- **scope.value**: The stream with wildcard (*) for device id or path components.
- **uploadInterval**: Fire if the amount of time between uploads exceeds this interval.
- **uploadTimeUnit**: The units of the **uploadInterval** value—**seconds**, **minutes**, **hours**.
- **readingInterval**: Fire if the amount of time between individual samples/reading exceeds this interval.
  readingInterval}} value—**seconds**, **minutes**, **hours**.
- **reset.disabled**: Resets automatically when data is reported, you can disable this by specifying **true**.

```
{
        "name":"Fleet sensor not reporting",
        "type":"Missing DataPoint",
        "scope":{
                "type":"Resource",
                "value":"*/drivestats/voltage"
        }
        "fire":{
                "parameters":{
                        "readingTimeUnit":"seconds",
                        "uploadTimeUnit":"minutes",
                        "uploadInterval":"1",
                        "readingInterval":"5"
                }
        },
        "reset": {
                "disabled": true
        }
}
```

Missing DiaChannel DataPoint

The following payload creates a **Missing DiaChannel DataPoint** alert. This alert type fires when the interval between data point uploads or data point measurements exceeds the requested time. Devices may cache data uploads, uploading infrequently (for example 24 hours), but uploading many measurements (for example, 24 total measurements if sampling every hour).

This alert automatically resets when data is reported.

Values:

- **scope.type**: One of **Group** or **Device**.
- **scope.value**: If type is **Group**, then value is the full group path (with no leading slash). If type is **Device**, then value is the full device ID. If type is **XBeeNode**, then the value is the XBee node extended **src** address.
- **uploadInterval**: Fire if the amount of time between uploads exceeds this interval.
- **uploadTimeUnit**: The units of the **uploadInterval** value—**seconds**, **minutes**, **hours**.
- **readingInterval**: Fire if the amount of time between individual samples/reading exceeds this

interval.
readingInterval}} value—**seconds**, **minutes**, **hours**.

- **channelName**: The dia channel name or wildcard (*) for all channels.
- **instanceName**: The dia channel instance name or wildcard (*) for all instances.
- **reset.disabled**: Resets automatically when data is reported, you can disable this by specifying **true**.

```
{
       "type": "Missing DiaChannel DataPoint",
       "scope": {
              "type": "Device",
              "value": "00000000-00000000-000000FF-FF000000"
       },
       "fire": {
              "parameters": {
                     "channelName": "chan1",
                     "instanceName": "*",
                     "readingInterval": "10",
                     "readingTimeUnit": "minutes",
                     "uploadInterval": "1",
                     "uploadTimeUnit": "hours"
              }
       }
}
```

Missing Smart Energy DataPoint

The following payload creates a **Missing Smart Energy DataPoint** alert. This alert type fires when the interval between data point uploads or data point measurements exceeds the requested time. Devices may cache data uploads, uploading infrequently (for example 24 hours), but uploading many measurements (for example, 24 total measurements if sampling every hour).

This alert automatically resets when data is reported.

Values:

- **scope.type**: One of **Group**, **Device**, or **XBeeNode**.
- **scope.value**: If type is **Group**, then value is the full group path (with no leading slash). If type is **Device**, then value is the full device ID. If type is **XBeeNode**, then the value is the XBee node extended **src** address.
- **uploadInterval**: Fire if the amount of time between uploads exceeds this interval.
- **uploadTimeUnit**: The units of the **uploadInterval** value—**seconds**, **minutes**, **hours**.
- **readingInterval**: Fire if the amount of time between individual samples/reading exceeds this interval.
  readingInterval}} value—**seconds**, **minutes**, **hours**.
- **attributeId**: The XBee attribute id.
- **clusterId**: The XBee cluster id.
- **clusterType**: The XBee cluster type.

- **endpointId**: The XBee endpoint id.
- **reset.disabled**: Resets automatically when data is reported, you can disable this by specifying **true**.

```
{
       "type": "Missing Smart Energy DataPoint",
       "scope": {
              "type": "Group",
              "value": "New York/Sensors"
       },
       "fire": {
              "parameters": {
                     "attributeId": "1",
                     "clusterId": "1794",
                     "clusterType": "0",
                     "endpointId": "1",
                     "readingInterval": "1",
                     "readingTimeUnit": "hours",
                     "uploadInterval": "24",
                     "uploadTimeUnit": "hours"
              }
       },
}
```

Smart energy data point condition match

The following payload creates a **Smart energy data point condition match** alert. This alert type fires when the value reported in a DIA channel matches the condition criteria.

Values:

- **scope.type**: One of **Group**, **Device**, or **XBeeNode**.
- **scope.value**: If type is **Group**, then value is the full group path (with no leading slash). If type is **Device**, then value is the full device ID. If type is **XBeeNode**, then the value is the XBee node extended **src** address.
- **timeout**: Fire or reset if the value matches for this amount of time.
- **timeUnit**: One of **seconds**, **minutes**, **hours**.
- **operator**: One of **<**, **<=**, **>**, **>=**, **=**, **!=**.
- **type**: One of **string** or **numeric**.
- **thresholdValue**: The target value for the condition.
- **attributeId**: The XBee attribute id.
- **clusterId**: The XBee cluster id.
- **clusterType**: The XBee cluster type.
- **endpointId**: The XBee endpoint id.

```
{
       "type": "Smart energy data point condition match",
```

```
        "scope": {
                "type": "Group",
                "value": ""
        },
        "fire": {
                "disabled": false,
                "parameters": {
                        "attributeId": "1",
                        "clusterId": "1794",
                        "clusterType": "0",
                        "endpointId": "1",
                        "thresholdValue": "50",
                        "timeUnit": "seconds",
                        "timeout": "1",
                        "type": "numeric"
                }
        },
        "reset": {
                "disabled": false,
                "parameters": {
                        "attributeId": "1",
                        "clusterId": "1794",
                        "clusterType": "0",
                        "endpointId": "1",
                        "thresholdValue": "49",
                        "timeUnit": "seconds",
                        "timeout": "1",
                        "type": "numeric"
                }
        }
}
```

Subscription Usage

The following payload creates a **Subscription Usage** alert. This alert type fires when the account uses 95 percent of the devices available for use based on the Digi Remote Manager subscription.

There are a wide variety of values for the parameters of this alert. It is recommended that an alert is created with the Digi Remote Manager Web UI, and you determine the acceptable values from a call to **GET ws/v1/alerts/inventory**.

Values:

- **scope.type**: Always **Global**.
- **scope.value**: Always empty string.
- **svcId**: The service identifier.
- **unit**: The unit being checked.
- **metric**: The metric for the subscription data.
- **thresholdValue**: The target value for the condition.

```
{
        "name":"Too many devices",
        "type":"Subscription Usage",
        "scope":{
```

```
                "type":"Global",
                "value":""
        },
        "fire":{
                "parameters":{
                        "svcId":"3",
                        "unit":"%",
                        "metric":"devices",
                        "thresholdValue":"90"
                }
        }
}
```

XBeeNode Excessive Deactivations

The following payload creates an **XBeeNode Excessive Deactivations** alert. This alert type fires when a device goes offline many times during a time period.

Values:

- **scope.type**: One of **Group**, **Device**, or **XBeeNode**.
- **scope.value**: If type is **Group**, then value is the full group path (with no leading slash). If type is **Device**, then value is the full device ID. If type is **XBeeNode**, then the value is the XBee node extended **src** address.
- **deactivationCount**: Fires if the device deactivates more than this many times in the **deactivationWindow**
- **deactivationWindow**: The period of time in minutes for which to count deactivations.
- **activationWindow**: Resets when the Xbee node activates and stays activated for this number of minutes.

```
{
        "type":"XBeeNode Excessive Deactivations",
        "scope":{
                "type":"XbeeNode",
                "value":"00:00:00:00:00:00:00:00"
        },
        "fire":{
                "parameters":{
                        "deactivationWindow":"5",
                        "deactivationCount":"1"
                }
        },
        "reset":{
                "parameters":{
                        "activationWindow":"1"
                }
        }
}
```

XBeeNode Offline

The following payload creates an **XBeeNode Offline** alert. This alert type fires when an XBee node is offline for a period of time.

Values:

- **scope.type**: One of **Group**, **Device**, or **XBeeNode**.
- **scope.value**: If type is **Group**, then value is the full group path (with no leading slash). If type is **Device**, then value is the full device ID. If type is **XBeeNode**, then the value is the XBee node extended **src** address.
- **reconnectWindowDuration**: Fires if the device does not reconnect within this number of minutes.
- **reset.disabled**: Is set to true to cause this alert to not auto-reset. The default is false.

```
{
        "type":"XBeeNode Offline",
        "scope":{
                "type":"Group",
                "value":"New York/Manhattan/Sensors"
        },
        "fire":{
                "parameters":{
                        "reconnectWindowDuration":"1"
                }
        }
}
```

v1/devices

Use the **devices** web service to create, modify, delete, or get devices. The API also includes options for getting data (channel), management, and metrics data streams.

URI

```
http://<hostname>/ws/v1/devices
```

Formats

HTTP method	Format	Description	Parameters
GET	/ws/v1/devices	Get summary of the device APIs.	None
GET	/ws/v1/devices/bulk	Get a list of all devices in CSV format.	group child_groups orderby tag type address name query
POST	/ws/v1/devices/bulk	Create a batch of users.	Update
GET	/ws/v1/devices/channel/{id}	Get a list of data channels for a device.	None
GET	/ws/v1/devices/channel/{id}/{name}	Get a specific data channel for a device.	None

HTTP method	Format	Description	Parameters
GET, POST, DELETE	/ws/v1/devices/inventory	List devices, create or modify devices, and delete devices.	group child_groups size cursor order orderby tag type address name query
GET, DELETE	/ws/v1/devices/inventory/{id}	Get or delete a specific device.	
PUT	/ws/v1/devices/inventory/{id}	Update a specific device.	update_only allow_swap
GET	/ws/v1/devices/management/{id}	Get a list of management streams for a device.	None
GET	/ws/v1/devices/management/{id}/{name}	Get a specific management stream for a device.	None
GET	/ws/v1/devices/metrics/{id}	Get a list of health metrics for a device.	None
GET	/ws/v1/devices/metrics/{id}/{name}	Get a specific health metric for a device.	None
GET	/ws/v1/devices/types	Get a list of device types.	None

Device fields

address

Device address supplied by the device.

alerts

Total number of fired alerts for the device.

capabilities

Capabilities to enable for the device (write-only):

Capability	Description
sm_compression_available	Allow compression to be used for SM requests: true or false.
sm_pack_available	Allow pack commands to be sent to the device (multiple commands in a single datagram): true or false.
sm_battery_operated	Send requests to the device in battery-operated mode: true or false.
sm_udp_enabled	Enable SM/UDP for the device: true or false.

channels_uri

Full path to channel data.

connection_status

Keyword that indicates the connection status of the device: connected or disconnected.

description

String that describes the device.

extended_address

XBee radio EUI64 extended address for the device.

firmware_version

String that indicates the firmware version of the device.

group

Group path for the device.

id

System-generated identifier for the device.

install_code

Installation code for the device. An installation code is required for any device manufactured with an associated installation code.

- If you attempt to add a device that requires an installation code with a missing or incorrect code, you receive an HTTP status 400 error code along with a message describing the error.
- If you are adding multiple devices and one or more of the device installation code is missing or incorrect, you receive an HTTP status 207 error along with a message describing the error.

ip

Local IP address of the device.

last_connect

Date and time the device last connected to Remote Manager in ISO 8601 format.

last_disconnect

Date and time the device last disconnected from Remote Manager in ISO 8601 format.

last_update

Date and time the device was last updated in ISO ISO 8601 format.

mac

MAC address of the device.

management_uri

Full path to management data.

metrics_uri

Full path to metrics data.

notes

String that provide notes for the device (formerly user meta data).

parent

System-generated identifier for the parent device.

password

Password for the device (write-only).

public_ip

Public IP address of device (formerly called global IP).

restricted_status

Keyword that indicates the restriction status of the device: unrestricted, disabled, restricted, or untrusted.

serial_number

Serial number of the device.

signal_percent

Percent of the primary cellular signal.

signal_percent2

Percent of the 2nd cellular signal.

signal_quality

Signal quality of the primary cellular signal.

signal_quality2

Signal quality of the 2nd cellular signal.

signal_strength

Signal strenth of the primary cellular signal.

signal_strenth2

Signal strength of the 2nd cellular signal.

tags

An array of tags associated with the device.

type

String that indicates the device type for the device.

vendor_id

Vendor identifier assigned to the device.

Channel, management, and metric fields

id

System-generated identifier for the device.

name

Name for the data channel.

value

Current value of the channel.

units

Units for the channel value.

timestamp

Time the current value was reported (ISO 8601 standard format).

history_uri

URI to the history of channel values (in streams).

customer_id

Identifier for the customer that owns the data.

Parameters

Name	Type	Description	Notes
tag	string	Query devices with matching tags.	(tag | type | address) Use only one per request.
type	string	Query devices by type.	
address	string	Query devices by address.	
cursor	string	Cursor to get the next page of devices. Omit on initial call.	
size	integer	Number of itmes to return. The maximum and default is 1000.	

Example: List all devices

The following example shows how to list all devices in your inventory.

Request

```
GET /ws/v1/devices/inventory
```

Response

```
{
  "count" : 2,
  "size" : 1000,
  "start" : 0,
  "total" : 2,
  "list" : [ {
    "mac" : "00:40:9D:29:8D:0E",
    "id" : "00000000-00000000-00409DFF-FF298D0E",
    "vendor_id" : 4261412864,
    "type" : "ConnectPort X8",
    "firmware_version" : "2.13.0.12",
    "restricted_status" : "unrestricted",
    "ip" : "10.20.1.68",
    "public_ip" : "10.20.1.68",
    "connection_status" : "disconnected",
    "last_connect" : "2014-03-11T14:33:35.480Z",
    "description" : "",
    "extended_address" : "00:13:A2:00:40:0A:3F:05",
    "tags" : [
        "gateway",
        "DIA"
    ],
    "group" : "Primary",
    "last_disconnect" : "2014-03-11T18:51:05.597Z"
  }, {
    "id" : "F9F967B4-33804DCE-2BDC4702-45053B1C",
    "vendor_id" : 4261412871,
    "type" : "DIA Device",
    "restricted_status" : "untrusted",
    "connection_status" : "disconnected",
    "description" : "Thermostat",
    "tags" : ["tstat"],
    "group" : "Primary",
    "address" : "dia/00000000-00000000-00409DFF-FF298D0E/sensor0",
    "parent" : "00000000-00000000-00409DFF-FF298D0E"
  },
  ...
 ]
}
```

Example: List all devices using query by tags

The following example shows how to list all devices that are tagged with a specific tag.

Request

```
GET /ws/v1/devices/inventory?query=tags='myTag'
```

Response

```
{
  "count" : 3,
  "size" : 1000,
  "list" : [ {
    "channels_uri" : "/ws/v1/devices/channels/8C1C080F-A8214448-2674ED16-
B9196C0E",
    "metrics_uri" : "/ws/v1/devices/metrics/8C1C080F-A8214448-2674ED16-B9196C0E",
    "group" : "",
    "management_uri" : "/ws/v1/devices/management/8C1C080F-A8214448-2674ED16-
B9196C0E",
    "type" : " ",
    "connection_status" : "disconnected",
    "id" : "8C1C080F-A8214448-2674ED16-B9196C0E",
    "restricted_status" : "unrestricted",
    "tags" : [ "myTag" ],
    "health_status" : "unknown",
    "maintenance_mode" : "off"
  }, {
    "channels_uri" : "/ws/v1/devices/channels/BD2EA7BE-60DF46A6-EBBF5052-
5B9217FC",
    "metrics_uri" : "/ws/v1/devices/metrics/BD2EA7BE-60DF46A6-EBBF5052-5B9217FC",
    "group" : "",
    "management_uri" : "/ws/v1/devices/management/BD2EA7BE-60DF46A6-EBBF5052-
5B9217FC",
    "type" : " ",
    "connection_status" : "disconnected",
    "id" : "BD2EA7BE-60DF46A6-EBBF5052-5B9217FC",
    "restricted_status" : "unrestricted",
    "tags" : [ "myTag" ],
    "health_status" : "unknown",
    "maintenance_mode" : "off"
  }, {
    "channels_uri" : "/ws/v1/devices/channels/BF652035-B3BA464A-B5C95C90-
770A4838",
    "metrics_uri" : "/ws/v1/devices/metrics/BF652035-B3BA464A-B5C95C90-770A4838",
    "group" : "",
    "management_uri" : "/ws/v1/devices/management/BF652035-B3BA464A-B5C95C90-
770A4838",
    "type" : " ",
    "connection_status" : "disconnected",
    "id" : "BF652035-B3BA464A-B5C95C90-770A4838",
    "restricted_status" : "unrestricted",
    "notes" : "yo",
    "tags" : [ "myTag" ],
    "health_status" : "unknown",
    "maintenance_mode" : "off"
  } ]
}
```

Example: Get a single device

The following example shows how to get details for a single device with an ID of F9F967B4-33804DCE-2BDC4702-45053B1C.

Request

```
GET /ws/v1/devices/inventory/F9F967B4-33804DCE-2BDC4702-45053B1C
```

Response

```
{
  "id" : "F9F967B4-33804DCE-2BDC4702-45053B1C",
  "vendor_id" : 4261412871,
  "type" : "DIA Device",
  "restricted_status" : "untrusted",
  "connection_status" : "disconnected",
  "description" : "Thermostat",
  "tags" : ["tstat"],
  "group" : "Primary",
  "address" : "dia/00000000-00000000-00409DFF-FF298D0E/sensor0",
  "parent" : "00000000-00000000-00409DFF-FF298D0E"
}
```

Example: Create a device

Use the following API to create or update one or more devices. If you don't provide a device ID, Remote Manager automatically assigns an ID and returns the device ID in the response.

The request payload can include a single device or multiple devices. Multiple devices must be wrapped in an array for JSON and a <list> element for XML. The response always returns the devices as a list.

The following sample JSON request creates a device with a type, a restricted_status, and tags. A device ID is not provided.

Request

```
POST /ws/v1/devices/inventory
```

```
{
  "type" : "Acme SuperDevice",
  "restricted_status" : "unrestricted",
  "tags" : ["excellent","awesome"]
}
```

Response

```
{
    "count": 1,
    "list": [
        {
            "connection_status": "disconnected",
            "id": "D12FC38D-4731448E-BA203D94-AF4FE23F",
            "restricted_status": "unrestricted",
            "tags": [
                    "excellent",
                    "awesome"
            ],
            "type": "Acme SuperDevice"
        }
    ]
}
```

Example: Create multiple devices

The following sample XML request creates two devices wrapped in a <list>:

Request

```
POST /ws/v1/devices/inventory
```

```
<list>
  <device>
    <type>Acme SuperDevice</type>
    <restricted_status>unrestricted</restricted_status>
    <tags>
        <tag>excellent</tag>
        <tag>awesome</tag>
    </tags>
  </device>
  <device>
    <id>12345</id>
    <type>Acme SuperDevice</type>
    <restricted_status>unrestricted</restricted_status>
    <tags>
        <tag>excellent</tag>
        <tag>awesome</tag>
    </tags>
  </device>
</list>
```

Response (XML)

```
<results>
  <count>2</count>
  <list>
    <device>
      <id>C35ABF3A-AB14468F-9ABDC4E4-12E8D89B</id>
      <type>Acme SuperDevice</type>
      <restricted_status>unrestricted</restricted_status>
      <connection_status>disconnected</connection_status>
      <tags>
        <tag>excellent</tag>
        <tag>awesome</tag>
      </tags>
    </device>
    <error>
      <error_status>400</error_status>
      <error_message>Identifier must have 4 segments and supplied device ID
&amp;quot;12345&amp;quot; has 1.</error_message>
    </error>
  </list>
</results>
```

Example: Edit a device

The following examples shows how to edit the tag list and restriction status for a device.

Request

```
PUT ws/v1/devices/inventory/F9F967B4-33804DCE-2BDC4702-45053B1C
```

```
{
  "tags" : ["bad"],
  "restricted_status" : "untrusted"
}
```

Response

```
 {
  "id": "F9F967B4-33804DCE-2BDC4702-45053B1C",
  "restricted_status": "untrusted",
  "tags": ["bad"]
}
```

Example: List device channels

The following example shows how to list the data channels reported by a device. A device can have up to 500 data channels.

Request

```
GET ws/v1/devices/channels/F9F967B4-33804DCE-2BDC4702-45053B1C.xml
```

Response (XML)

```
<results>
  <count>3</count>
  <list>
    <channel>
      <id>F9F967B4-33804DCE-2BDC4702-45053B1C</id>
      <name>temperature</name>
      <value>71</value>
      <units>F</units>
      <timestamp>2014-02-23T19:12:57.283Z</timestamp>
      <history_uri>/ws/v1/streams/history/F9F967B4-33804DCE-2BDC4702-
45053B1C/temperature</history_uri>
    </channel>
    <channel>
      <id>F9F967B4-33804DCE-2BDC4702-45053B1C</id>
      <name>light</name>
      <value>10</value>
      <timestamp>2014-02-23T19:12:57.983Z</timestamp>
      <history_uri>/ws/v1/streams/history/F9F967B4-33804DCE-2BDC4702-
45053B1C/light</history_uri>
    </channel>
    <channel>
      <id>F9F967B4-33804DCE-2BDC4702-45053B1C</id>
      <name>humidity</name>
      <value>44</value>
      <units>%</units>
      <timestamp>2014-02-23T19:12:56.510Z</timestamp>
      <history_uri>/ws/v1/streams/history/F9F967B4-33804DCE-2BDC4702-
45053B1C/humidity</history_uri>
    </channel>
  </list>
</results>
```

Example: Delete a device

The following example shows how to delete a device from your inventory.

```
DELETE /ws/v1/devices/inventory/F9F967B4-33804DCE-2BDC4702-45053B1C
```

No response is returned unless there is an error.

v1/events

Use the **v1/events** web service to retrieve events.

URI

```
http://<hostname>/ws/v1/events
```

Formats

HTTP method	Format	Description	Parameters
GET	/ws/v1/events	Get a summary of the events APIs.	None
GET	/ws/v1/events/bulk	Retrieve events for the current user account in CSV format.	start_time end_time
GET	/ws/v1/events/inventory	Retrieve events for the current user account.	cursor size start_time end_time

v1/groups

Use the **v1/groups** web service to create, update, list, or remove groups.

URI

```
http://<hostname>/ws/v1/groups
```

Formats

HTTP method	Format	Description	Parameters
GET	/ws/v1/groups	Get summary of the groups APIs.	None
GET	/ws/v1/groups/inventory	Get a list of groups.	orderby query
POST	/ws/v1/groups/inventory/path	Create a new group.	path description
PUT	/ws/v1/groups/inventory/path	Change the name of a group.	None
DELETE	/ws/v1/groups/inventory/path	Remove one or more groups and subgroubs.	move_devices remove_ subgroups

Parameters

path

Full path for the group, including the group name.

description

Text description of the group.

move_devices

Specify whether to remove devices in the group being deleted. If you do not remove devices, any existing devices within the removed group are moved to the root group. If you delete devices, devices in removed groups are removed from your device inventory. The default is no.Text description of the group.

v1/health_configs

Use the health_configs API to get a list of health configurations for your account, as well as modify or delete a health configuration.

URI

```
http://<hostname>/ws/v1/health_configs
```

Formats

HTTP Method	Format	Description
GET	/ws/v1/health_configs	Get a summary of the health_configs APIs.
GET	/ws/v1/health_configs/inventory	Get a list of all health configurations in your inventory.
GET, PUT, DELETE	/ws/v1/health_configs/inventory/{id}	Get, modify, or delete a named health configuration.

Fields

id

System-generated identifier for the device type to which the health_config applies.

Parameters

Name	Type	Description
cursor	string	Cursor to get the next page of health configurations. Omit on initial call.
size	integer	Number of items to return. The maximum and default is 1000.
replace	boolean	Boolean value that indicates whether to replace the current report configuration. **True** or **Yes**: Replace the current report configuration. **False** or **No**: Update the existing report configuration with changes.

Example: Get a summary of the health_config API

The following example shows how to get a summary of the health_config API.

Request

```
/ws/v1/health_configs
```

Response

```
{
  "count" : 3,
  "list" : [ {
    "path" : "/ws/v1/health_configs",
    "requests" : [ {
      "method" : "GET"
    } ]
  }, {
    "path" : "/ws/v1/health_configs/inventory",
    "requests" : [ {
      "method" : "GET",
      "params" : [ "size", "cursor" ]
    } ]
  }, {
    "path" : "/ws/v1/health_configs/inventory/{id}",
    "requests" : [ {
      "method" : "GET"
    }, {
      "method" : "PUT",
      "params" : [ "replace" ]
    }, {
      "method" : "DELETE"
    } ]
  } ]
}
```

Example: Get a list of health configurations for your account

The following example shows how to get a list of all health configurations for your account.

Request

```
/ws/v1/health_configs/inventory
```

Response

Note Because each device health configuration defines thresholds for multiple health metrics, the output is not shown here. To see sample output, go to **Documentation > API Explorer** and send the **v1/health_configs/List all health configs** to see details for all health configurations and corresponding metrics on your system.

Example: Get a specific health configuration in XML

The following example shows how to get the health configuration FE000002/TRANSPORT WR21 in XML format:

```
/ws/v1/health_configs/inventory/FE000002/TRANSPORT WR21.xml
```

Example: Disable a health configuration

Using the HTTP PUT method, the following example shows how to disable the health configuration FE000002/TRANSPORT WR21.

Request

```
/ws/v1/health_configs/inventory/FE000002/TRANSPORT WR21
```

```
{
  "enabled": false,
}
```

Example: change a health configuration

The following example shows how to change an existing Transport WR11 health configuration.

Request

```
/ws/v1/health_configs/inventory//FE000002/TRANSPORT WR11
```

```
{
  "id": "FE000002/TRANSPORT WR11",
  "enabled": true,
  "rules":[
    {
      "enabled":true,
      "stream":"DataPoint/*/metrics/sys/mem/free",
      "threshold":{
        "error":{
          "ranges":[
            {
              "upper":150000
            }
          ]
        },
        "warning":{
          "ranges":[
            {
              "lower":150000,
              "upper":700000,
              "lower_bound_type":"OPEN",
              "upper_bound_type":"OPEN"
            }
          ]
        },
        "normal":{
          "ranges":[
            {
              "lower":700000
            }
          ]
        }
      }
    }
  ]
}
```

Response

Note Because each device health configuration defines thresholds for multiple health metrics, the output is not shown here. To see sample output, go to **Documentation > API Explorer** and send the **v1/health_configs/Change a health config** to see sample results.

v1/jobs

List, cancel, or delete jobs in an account.

URI

```
http://<hostname>/ws/v1/jobs
```

Formats

Method	Formats	Description
GET	/ws/v1/jobs	Get a summary of the jobs API.
GET	/ws/v1/jobs/inventory	Get a list of all jobs.
GET	/ws/v1/jobs/inventory.xml	Get a list of all jobs in XML format.
GET	/ws/v1/jobs/bulk	Retrieve a list of jobs in CSV format.
GET	/ws/v1/jobs/{job_id}	Retrieve a job
PUT	/ws/v1/jobs/cancel{job_id}	Cancel one or more jobs.
PUT	/ws/v1/jobs/inventory/cancel?query=username='{username}'	Cancel one or more jobs for a user.
DELETE	/ws/v1/jobs/inventory/{id}	Delete a job.
DELETE	/ws/v1/jobs/cancel?query=job_type='{job_type}'	Delete one or more jobs by job type.

Parameters

Name	Description
orderby	Specify any field described in the query parameter syntax. Optionally add 'asc' or 'desc' to control the sort order. For example, to order with most recently created jobs first, specify orderby=id desc. **Note** The default sort order is desc (descending).
query	Specify the jobs query to evaluate. See Query language for v1 APIs.
cursor	
size	

Query fields

- carrier/carrier2—the current provider of the primary or secondary cellular service
- connection_status—one of 'connected' or 'disconnected'
- contact—any user contact information

- description—any description associated with the device
- firmware_version—the firmware level
- health_status—one of 'normal', 'warning', 'error' or 'unknown'
- id—the device ID
- ip—the last known IP address of the device
- last_connect—last connect time of the device
- last_disconnect—last disconnect time of the device
- last_update—last update time of the device
- location—the device location
- mac - the MAC address
- name - the device name
- network/network2—the current network (for example LTE) of the primary or secondary cellular service
- notes—device notes, also sometimes referred to as user meta data)
- public_ip—the last known global IP address of the device
- restricted_status—one of 'unrestricted', 'restricted', 'disabled', 'untrusted'
- serial_number—the device serial number
- signal_percent/signal_percent2—the percent of signal strength from 0 to 100 primary or secondary cellular service
- signal_quality/signal_quality2—the signal quality of the primary or secondary cellular service
- signal_strength/signal_strength2—the signal strength of the primary or secondary cellular service
- type—the device type
- vendor_id—the vendor ID value of the device

Query operators

- =, <>—Equality comparisons, used on any numeric, group, tag, text or enumerated fields
- <, <=, >, >=—Relative comparisons, used on any numeric or text fields. Not used for group, tag or enumerated fields
- startsWith, endsWith, contains—Used on any group, tag or text matching fields. Not used for numeric fields

Timestamp field comparisons are not supported.

Query examples

- Complex queries

```
query=group startsWith '/NorthWest' and (connection_status =
'disconnected' or signal_percent < 20)
```

  Find any devices in the /Northwest group and any subgroups that are either disconnected or have a low signal strength.

```
query=tags = 'important' and (health_status = 'error' or health_status
= 'warning')
```

Find any devices that have the 'important' tag and are in an error or warning health status.

- Group queries

```
query=group = '/test'
```

Query full group path, so matches any device in group '/test' and ignores any subgroups.

```
query=group startsWith 'test/'
```

Query full group path, so matches any device in the test root group and any subgroups.

```
query=group startsWith 'test'
```

Query full group path, so matches any device in any root group whose name starts with 'test' and all subgroups.

```
query=group endsWith '/leaf'
```

Query full group path, so matches any device in any path that ends with the group name 'leaf'.

- Tag Queries

```
query=tags = 'sensor'
```

Matches any device having a tag 'sensor'.

```
query=tags <> 'sensor'
```

Matches any device having no tag 'sensor'.

```
query=tags contains 'ns'
```

Matches any device having any tag containing 'ns'.

```
query=tags startsWith 'sens'
```

Matches any device having any tag that starts with 'sens'.

```
query=tags endsWith 'or'
```

Matches any device having any tag that ends with 'or'.

v1/metadata

Use the **v1/metadata** web service to retrieve device descriptors.

URI

```
https://<hostname>/ws/v1/metadata
```

Formats

HTTP method	Format	Description	Parameters
GET	/ws/v1/metadata	Get a summary of the /ws/v1/metatdata APIs.	
GET	/ws/v1/metadata/device/descriptors/settings/{vendorId}/{deviceType}/{firmwareVersion:.}	Retrieves (GET) the query_settings descriptor data for a vendor, device type/firmware version.	product_id firmware_id
GET	/ws/v1/metadata/device/descriptors/state/{vendorId}/{deviceType}/{firmwareVersion:.}	Retrieves (GET) the query_state descriptor data for a vendor, device type/firmware version.	product_id firmware_id
GET	/ws/v1/metadata/device/descriptors/ui/{vendorId}/{deviceType}/{firmwareVersion:.}	Retrieves (GET) ui descriptor data for a vendor, device type/firmware version.	product_id firmware_id allow_fallback

v1/monitors/history

Use the v1/monitors/history API to retrieve saved push notifications that may have been sent or have yet to be sent to push monitors. This feature is available for customers who are subscribed to the Push Monitor service.

Not all monitors save push notifications. Monitors that support saving of push notifications (sometimes called persistent monitors) can have those notifications replayed on restart and the history can be retrieved. When a monitor is updated to change the topics that are being monitored, existing saved push notifications are not affected. Regardless of when the events are generated, event timestamps are saved using the server timestamp. Queries using this API with start_time or end_time parameters restrict the returned events based on the server timestamps of those events. Queries using this API with a cursor or when receiving more than one push notification, enable the return of a polling cursor.

Polling cursor

In addition to the normal paged output options for version 1 web services (for example, count, size, cursor and next_uri), the monitors/history API supports polling for subsequent results by returning a polling cursor (result elements polling_cursor and polling_uri). The polling cursor is returned for every monitors/history query that has returned any results or has received a cursor as input, allowing a continuation of that query at a later time.

You can always use a polling_cursor to poll for any added push notifications (which are ordered by timestamp) that were saved by the monitor since the last call that returned the polling_cursor. If the API is paging through a large set of existing results, the polling cursor and the traditional cursor are both be set to an identical value. However, if all results have been returned for a continued query, no

traditional cursor is set because all pages of existing results have been returned. Those results will still receive a polling_cursor element, allowing further continuation of those queries. The polling_cursor avoids having to manually calculate the required timestamp and add the start_time parameter to subsequent queries: use the polling_cursor or the polling_uri directly to retrieve subsequent data.

URI

```
http://<hostname>/ws/v1/monitors/history
```

Formats

Method	Formats	Description
GET	/ws/v1/monitors/history	Get a list of saved push notifications.

Parameters

Name	Type	Description
start_time	timestamp	Specify the start time (inclusive) in ISO 8601 or epoch (long). The default value is the first saved notification.
end_time	timestamp	Specify the end time (exclusive) in ISO 8601 or epoch (long). The default value is the current last saved notification.
size	integer	Specify the maximum number of saved push notifications to return.
cursor	string	Cursor to get the next page of devices. Omit on initial call.
polling_cursor	string	Returned for every monitors/history query that has returned any results or has received a cursor as input, allowing a continuation of the query at a later time.
next_uri	string	URI value that can be used to request the next page of data. No value is returned if there are no more pages available.
polling_uri	string	Returned for every monitors/history query that has returned any results.

Example: Query polling monitor history

The following example shows how to get all event history for polling monitor 433016.

Request

```
/ws/v1/monitors/history/433016
```

Response

```
{
    "count": 1,
    "list": [
        {
            "DataPoint": {
                "cstId": 73846,
                "data": "0",
                "description": "",
                "id": "80ba3970-563d-11e5-b865-fa163ed14178",
                "quality": 99,
                "serverTimestamp": 1441725785735,
                "streamId": "00000000-00000000-00409DFF-FF50B8B1/temp",
                "streamType": "FLOAT",
                "streamUnits": "Kelvin",
                "timestamp": 1441725785735
            },
            "group": "*",
            "operation": "INSERTION",
            "timestamp": "2015-09-08T15:23:05.888Z",
            "topic": "73846/DataPoint/00000000-00000000-00409DFF-FF50B8B1/temp"
        }
    ],
    "polling_cursor": "80d1920a-563d-11e5-b865-fa163ed14178",
    "polling_uri": "/ws/v1/monitors/history/440495?cursor=80d1920a-563d-11e5-
b865-fa163ed14178",
    "size": 1000
}
```

v1/settings

Use the **settings** web service to create, update, or list device settings for your account.

URI

```
http://<hostname>/ws/v1/settings
```

Formats

HTTP method	Format	Description	Parameters
GET	/ws/v1/settings	Get summary of the settings APIs.	

HTTP method	Format	Description	Parameters
GET	/ws/v1/settings/inventory /ws/v1/settings/inventory.xml	Get a list of all settings for your account.	
GET	/ws/v1/settings/inventory/{name}	Get the value for a specific setting.	name
PUT	/ws/v1/settings/inventory/	Change the value for a setting.	name value
Delete	/ws/v1/groups/inventory/{name}	Remove a setting.	name

v1/reports

Remote Manager generates status reports for alarms, connections, health, and monitors. Use the reports API to get a list of all available reports or a specific status report.

URI

```
http://<hostname>/ws/v1/reports
```

Formats

Method	Formats	Description
GET	/ws/v1/reports	Get a list of available reports, including summary information.
GET	/ws/v1/reports/alarms	Get a list of fired alarms, along with the total count for each alarm type.
GET	/ws/v1/reports/alerts	Get a list of all fired alerts, along with the total count for each alert type.
GET	/ws/v1/reports/connections	Get a connection status summary for all devices by state: connected, disconnected, never connected.
GET	ws/v1/reports/connections/history	Get a history of aggregate connection status for all devices by state: connected, disconnected, never connected. Use start_time and end_time to specify the range of data to include.

Method	Formats	Description
GET	/ws/v1/reports/devices	Get a report based on a device query: /ws/v1/reports/devices/carrier—Summarize current device network providers. /ws/v1/reports/devices/carrier2—Summarize current device network providers. /ws/v1/reports/devices/network—Summarize current device networks. /ws/v1/reports/devices/signal_percent—Summarize current device signal. /ws/v1/reports/devices/connection_status—Summarize current device connections. /ws/v1/reports/devices/health_status—Summarize current device health. /ws/v1/reports/devices/restricted_status—Summarize current device restricted status. /ws/v1/reports/devices/vendor_id—Summarize current device vendor IDs. /ws/v1/reports/devices/type—Summarize current device types. /ws/v1/reports/devices/firmware_version—Summarize current device firmware versions. /ws/v1/reports/devices/geolocation—Get device geolocation.
GET	ws/v1/reports/health_status	Get a summary of health status for all devices, along with counts for each health metric.
GET	/ws/v1/reports/health	Get a list of devices in each health state: normal, warning, error, and unknown.
GET	/ws/v1/reports/monitors	Get a list of monitor states for the account, along with the number of entities in each state: inactive, active, suspended, and disconnecting.

Parameters

Name	Type	Description
child_groups	string	Boolean value that specifies whether to include all children of the group in the status: true to include all child groups; false to include only the parent group. The default is true.

Name	Type	Description
end_time	timestamp	Specify the end time (exclusive) in ISO 8601 or epoch (long).
group	string	Specify a group to get status information for the group.
query	string	Specify the device query to summarize. The query language is similar to SQL Query sntax: ■ SQL-like conditions using AND, OR, and parenthesis to group expressions ■ Various condition operators on numeric and text values ■ Single quoted text literals: 'TheText' ■ Text escape for quote character is the quote: 'isn''t difficult' ■ Numeric literals support 0x prefix for hex ■ Enumerated values (like connection_status) are treated as text ■ Case insensitive comparisons Timestamp field conditions are not supported.
start_time	timestamp	Specify the start time (inclusive) in ISO 8601 or epoch (long).
scope	string	Specify **primary** or **secondary** for any of the **carrier**, **network**, and **signal_percent** device reports. The report summarizes information for the primary or secondary cellular modem. The default is **primary**.
type	string	Specify a device type for which you want to get status.

Query fields

- carrier/carrier2—the current provider of the primary or secondary cellular service
- connection_status—one of 'connected' or 'disconnected'
- contact—any user contact information
- description—any description associated with the device
- firmware_version—the firmware level
- health_status—one of 'normal', 'warning', 'error' or 'unknown'
- id—the device ID
- ip—the last known IP address of the device
- last_connect—last connect time of the device

- last_disconnect—last disconnect time of the device
- last_update—last update time of the device
- location—the device location
- mac - the MAC address
- name - the device name
- network/network2—the current network (for example LTE) of the primary or secondary cellular service
- notes—device notes, also sometimes referred to as user meta data)
- public_ip—the last known global IP address of the device
- restricted_status—one of 'unrestricted', 'restricted', 'disabled', 'untrusted'
- serial_number—the device serial number
- signal_percent/signal_percent2—the percent of signal strength from 0 to 100 primary or secondary cellular service
- signal_quality/signal_quality2—the signal quality of the primary or secondary cellular service
- signal_strength/signal_strength2—the signal strength of the primary or secondary cellular service
- type—the device type
- vendor_id—the vendor ID value of the device

Query operators

- =, <>—Equality comparisons, used on any numeric, group, tag, text or enumerated fields
- <, <=, >, >=—Relative comparisons, used on any numeric or text fields. Not used for group, tag or enumerated fields
- startsWith, endsWith, contains—Used on any group, tag or text matching fields. Not used for numeric fields

Timestamp field comparisons are not supported.

Query examples

- Complex queries

```
query=group startsWith '/NorthWest' and (connection_status =
'disconnected' or signal_percent < 20)
```

Find any devices in the /Northwest group and any subgroups that are either disconnected or have a low signal strength.

```
query=tags = 'important' and (health_status = 'error' or health_status
= 'warning')
```

Find any devices that have the 'important' tag and are in an error or warning health status.

- Group queries

```
query=group = '/test'
```

Query full group path, so matches any device in group '/test' and ignores any subgroups.

```
query=group startsWith 'test/'
```

Query full group path, so matches any device in the test root group and any subgroups.

```
query=group startsWith 'test'
```

Query full group path, so matches any device in any root group whose name starts with 'test' and all subgroups.

```
query=group endsWith '/leaf'
```

Query full group path, so matches any device in any path that ends with the group name 'leaf'.

- Tag Queries

```
query=tags = 'sensor'
```

Matches any device having a tag 'sensor'.

```
query=tags <> 'sensor'
```

Matches any device having no tag 'sensor'.

```
query=tags contains 'ns'
```

Matches any device having any tag containing 'ns'.

```
query=tags startsWith 'sens'
```

Matches any device having any tag that starts with 'sens'.

```
query=tags endsWith 'or'
```

Matches any device having any tag that ends with 'or'.

Example: Get a summary of the reports API

The following examples gets a summary of the reports API.

Request

```
GET /ws/v1/reports
```

Response

```
{
  "count" : 5,
  "list" : [ {
    "path" : "/ws/v1/reports",
    "requests" : [ {
      "method" : "GET"
    } ]
  }, {
    "path" : "/ws/v1/reports/alarms",
    "requests" : [ {
      "method" : "GET"
    } ]
  }, {
    "path" : "/ws/v1/reports/connections",
    "requests" : [ {
      "method" : "GET",
      "params" : [ "group", "type", "child_groups" ]
    } ]
  }, {
```

Example: Get a report of fired alarms

The following example shows how to get a report on fired alarms that includes totals for each alarm type:

Request

```
GET /ws/v1/reports/alarms
```

Response

```
{
  "count" : 8,
  "size" : 1000,
  "list" : [ {
    "id" : 11673,
    "name" : "System Throttles",
    "description" : "Detects when system alarm conditions occur for System
Throttles",
    "fired" : 1
  }, {
    "id" : 11971,
    "name" : "System Monitors",
    "description" : "Detects when system alarm conditions occur for System
Monitors",
    "fired" : 4
  }, {
    "id" : 14448,
    "name" : "Device Offline",
    "description" : "Detects when a device disconnects from Remote Manager and
fails to reconnected within the specified time",
    "fired" : 1
  }, {
    "id" : 15662,
    "name" : "Device Excessive Disconnects",
    "description" : "Detects devices with an excessive number of disconnects.",
    "fired" : 178
  }, {
    "id" : 15665,
    "name" : "Subscription Usage",
    "description" : "Fires when subscription usage exceeds a certain threshold",
    "fired" : 1
  }, {
    "id" : 16091,
    "name" : "Missing Smart Energy DataPoint",
    "description" : "Fires when devices have not reported Smart Energy data
within the specified time",
    "fired" : 1
  }, {
    "id" : 17141,
    "name" : "Subscription Usage",
    "description" : "Fires when subscription usage exceeds a certain threshold",
    "fired" : 1
  }, {
    "id" : 17163,
    "name" : "Missing DataPoint",
```

```
    "description" : "Fires when data points are not reported within the specified
time",
    "fired" : 1
  } ]
}
```

Example: Get a health status report

The following example shows how to get a health status report for all devices in your account:

Request

```
GET /ws/v1/reports/health
```

Response

```
{
  "unknown" : {
    "count" : 2
  },
  "normal" : {
    "count" : 2462
  },
  "warning" : {
    "count" : 5
  },
  "error" : {
    "count" : 3
  }
}
```

Example: Get connection status history report

The following example shows how to get a connection status history report.

Request

```
/ws/v1/reports/connections/history?start_time=2015-09-08T17:00:00.000Z&end_
time=2015-09-08T17:15:00.000Z
```

Response

```
[ {
  "connected" : 3,
  "disconnected" : 1,
  "never_connected" : 2,
  "start_time" : "2015-09-08T17:00:00.000Z"
}, {
  "connected" : 3,
  "disconnected" : 1,
  "never_connected" : 2,
  "start_time" : "2015-09-08T17:05:00.000Z"
}, {
  "connected" : 3,
  "disconnected" : 1,
  "never_connected" : 2,
  "start_time" : "2015-09-08T17:10:00.000Z"
}, {
  "connected" : 3,
  "disconnected" : 1,
```

```
  "never_connected" : 2,
  "start_time" : "2015-09-08T17:15:00.000Z"
} ]
```

Example: Get monitor status report

The following example shows how to get a monitor status report. The report lists the total number of monitors that are inactive, active, suspended, disconnecting, disabled, and connection.

Request

```
GET /ws/v1/reports/monitors
```

Response

```
{
  "inactive" : {
    "count" : 10
  },
  "active" : {
    "count" : 290
  },
  "suspended" : {
    "count" : 320
  },
  "disconnecting" : {
    "count" : 15
  },
  "disabled" : {
    "count" : 5
  },
  "connecting" : {
    "count" : 7
  }
}
```

v1/streams

Use the streams web service to manage data streams and data points. You can also use the streams web service to upload a batch of data points to streams using an XML or CVS file. See Direct device uploads.

URI

```
http://<hostname>/ws/v1/streams
```

Formats

Method	Format	Description	Parameters
GET	/ws/v1/streams	Get a summary of the streams APIs.	None
GET	/ws/v1/streams/bulk/history	Get historical data in CSV format.	order start_time end_time timeline
GET, POST	/ws/v1/streams/inventory	List, create, or modify data streams.	order cursor size category
GET	/ws/v1/streams/inventory/{stream_id}	Get a specific data stream.	order cursor, size start_time end_time timezone interval method
PUT, DELETE	/ws/v1/streams/inventory/{stream_id}	Create, modify, or delete a specific data stream.	
GET	/ws/v1/streams/inventory?category=carrier	Get carrier usage data for devices.	

Method	Format	Description	Parameters
POST	/ws/v1/streams/history	Add one or more data points to a data stream	
GET	/ws/v1/streams/history/{stream_id}	Get the history for a data stream.	order cursor size start_time end_time timeline
DELETE	/ws/v1/streams/history/{stream_id}	Delete the history for a data stream.	start_time, end_time, timeline
GET	/ws/v1/streams/rollups/{stream_id}	Get roll-up information for a data stream.	
GET	/ws/v1/streams/history/{device_id}/carrier/{sim_id}/usage/{usage_id}	Get carrier usage data for a device.	
GET	/ws/v1/streams/inventory?category=data	Get data streams reported by devices.	
GET	/ws/v1/streams/inventory?category=metrics	Get health metrics streams reported by devices.	
GET	/ws/v1/streams/inventory?category=management	Get management streams recorded for devices.	

Stream fields

id

Steam identifier.

description

Stream description.

type

Data type of the stream:

- integer
- long
- float
- double
- string
- binary

value

Current value of the data stream.

timestamp

Date and time the current value was set.

server_timestamp

Date and time the current value was received.

stream_units

Units for the data.

forward_to

List of additional streams to forward data to when received.

History fields

id

Identifier of the data point in the stream history.

stream_id

Stream identifier of the history data.

Roll-up fields

stream_id

Stream identifier for the roll-up data.

Parameters

Name	Type	Description
start_time	timestamp	Start time (inclusive) in ISO 8601 or epoch (long).
end_time	timestamp	End time (exclusive) in ISO 8601 or epoch (long).
timeline	string	Timestamps to use in the request: client or server. The default is client.
cursor	string	Cursor to get the next page of devices. Omit on initial call.
size	integer	Number of items to return. The maximum and default is 1000.
order	string	Return streams ordered by ID (asc | desc). The default is ascending (asc).
category	string	Return streams for the specified category: data, metrics, management, or carrier. If you do not use the category parameter, streams for all categories are returned.
timezone	string	Timezone in which to calculate rollups. Applies only to rollups with intervals of day or longer.
interval	string	Rollup interval: half, hour, day, week, or month. The default is hour.
method	string	Rollup method: sum, average, min, max, count, standarddev. The default is average.

Direct device uploads

Devices can upload directly to data streams over any of the existing transports (TCP, UDP, SMS, and Satellite). The path specified in the data service message begins with **DataPoint** and the rest of the message is mapped to a data stream appended to the device ID.

For example, if the device sends a data point file specifying the filename **DataPoint/temp1**, the data point is added to the data stream **<device-id>/temp1**. The file must follow one of the expected formats and must specify the format via the file extension. The following types are supported for a given extension:

Format	Extension	Description
XML	.xml	XML representation same as the /ws/DataPoint interface.
CSV	.csv	Comma separated list. One data point per line with details separated by commas.
Binary	.bin	Whatever the content of the uploaded data is directly inserted to a single data point.

Data limits related to direct device uploads

To maximize the speed and throughput of Remote Manager, limitations have been imposed on device uploads.

- Maximum number of data points allowed per request: 250
- Maximum size of Send Data requests: 2MB
- Maximum size of replies to Device Requests: 2MB
- Maximum number of binary data points allowed: 64KB

Note The Description field for a data point does not display in the Remote Manager UI Data Streams view.

When devices push data points up to Remote Manager, the description included refers to the data point, not the data stream. To view the description, you must retrieve data point via web services.

XML

XML format uses the same format used in /ws/DataPoint PUT. The stream id is ignored since it is provided by the path. Also, any streams listed in the **forwardTo** field will be normalized to the device's stream. This is done to prevent one device from uploading data into another device's stream.

```
<DataPoint>
    <data>42</data>
    <!-- Everything below this is optional -->
    <description>Temperature at device 1</description>
    <location>0.0, 0.0, 0.0</location>
    <quality>99</quality>
    <dataType>float</dataType>
    <units>Kelvin</units>
</DataPoint>
```

For multiple data points in one message:

```
<list>
  <DataPoint>
    <data>42</data>
    <timestamp>1234566</timestamp>
  </DataPoint>
  <DataPoint>
    <data>43</data>
  </DataPoint>
</list>
```

CSV

An optional upload format is to specify the data in UTF-8 encoded comma separated values. Each line (`'\n'` terminated) specifies a data point. The default order is:

```
DATA, TIMESTAMP, QUALITY, DESCRIPTION, LOCATION, DATATYPE, UNITS, FORWARDTO
```

Meaning the following file:

```
data, 1,99,"my description",,INTEGER,kelvins,"stream1,stream2"
data2,2,50,"my description"
data3,3,25,"my description"
```

Would create 3 data points, set the stream's units/type to kelvins/Integers, and have the data points with the data "data", "data2", and "data3", using the epoch timestamps of 1, 2, and 3.

Note that location was omitted in the above example. You can omit values by leaving them empty or stopping before the end. For example:

Empty values:data,1,,,99
Ending early:data,1

Order can be overridden. You can define a header on the first line by starting it with a '#' character, for example:

```
#TIMESTAMP,DATA
1, data
2, data2
3, data3
```

Will create 3 data points 1ms apart starting at epoch (1970).

Multiple datapoints for multiple streams from a device can be inserted in one message using the STREAMID value. When the STREAMID value is specified, the file name is no longer used for the stream name.

For example:

```
#STREAMID,DATA,TIMESTAMP
sensor1/port1,97,1
sensor1/port2,98,1
sensor2/port1,42,1
sensor2/port2,0,2
```

Will create 4 data points, one in each of 4 separate streams for the device. The first 3 data points are at 1ms after the epoch (1970) and the final data point is 1ms later.

The XML version is as follows:

```
<list>
<DataPoint><streamId>sensor1/port1</streamId><data>97</data><timestamp>1</timesta
mp></DataPoint>
<DataPoint><streamId>sensor1/port2</streamId><data>98</data><timestamp>1</timesta
mp></DataPoint>
<DataPoint><streamId>sensor2/port1</streamId><data>42</data><timestamp>1</timesta
mp></DataPoint>
<DataPoint><streamId>sensor2/port2</streamId><data>0</data><timestamp>2</timestam
p></DataPoint>
</list>
```

Binary Concise Alternative Format

The disadvantage to using the XML format is that it is very verbose. This binary alternative format can be used to be more concise. You can specify a simple value instead of XML or CSV data. When the value is pushed to /DataPoint, it is stored in complete as-is in time-series data (in the exact binary format as provided). For details on endianness, bit lengths, and so on for supported data types see the dataType in the Data Streams section. However, data types are not required. Data can be 1 byte status indicators or 10k images but Remote Manager will not be able to provide rollups on things which do not use the specified formats.

For instance, the following data service message:

path: /DataPoint/temp1.bin

content: 42

Will result in a new data point with a value of "42" (in binary).

Note: The binary concise mechanism has the following limitations:

- Only single values can be uploaded per data service message
- Data must be smaller than 64k

Deciding which format to use when inserting data

Whitespace characters for the data value are preserved in all formats. Use quotes around the string for CSV format to preserve break lines. For binary data, we recommend you to use binary concise format. Binary concise format however can't be used to create multiple data points in a single request. To create multiple binary data points in a single request, we recommend you to use a base64 encoded string.

Example: List all streams

Use the following API to list all streams for the current user.

Request

```
GET /ws/v1/streams/inventory
```

Response

```
{
    "count": 1000,
    "size": 1000,
    "cursor": "380a2605-392b-d5aa-392b-d5a9ad4571a0",
    "next_uri": "/ws/v1/streams/inventory?size=3&cursor=380a2605-392b-d5aa-392b-
d5a9ad4571a0",
    "list": [
        {
            "history_uri": "/ws/v1/streams/history/F9F967B4-33804DCE-2BDC4702-
45053B1C/humidity",
            "id": "F9F967B4-33804DCE-2BDC4702-45053B1C/humidity",
            "server_timestamp": "2014-02-23T19:12:59.847Z",
            "timestamp": "2014-02-23T19:12:56.510Z",
            "type": "LONG",
            "units": "%",
            "value": "44"
        },
        {
            "description": "freezer",
            "history_uri": "/ws/v1/streams/history/F9F967B4-33804DCE-2BDC4702-
45053B1C/temperature",
            "id": "F9F967B4-33804DCE-2BDC4702-45053B1C/temperature",
            "server_timestamp": "2014-02-23T19:12:59.848Z",
            "timestamp": "2014-02-23T19:12:57.283Z",
            "type": "LONG",
            "units": "F",
            "value": "71"
        },
        ...
    ],
}
```

Example: Get a stream

Use the following request to get a stream.

Request

```
GET /ws/v1/streams/inventory/F9F967B4-33804DCE-2BDC4702-45053B1C/temperature
```

Response

```
<results>
    <stream>
        <id>F9F967B4-33804DCE-2BDC4702-45053B1C/temperature</id>
        <description>freezer</description>
        <type>LONG</type>
        <value>71</value>
        <units>F</units>
        <timestamp>2014-02-23T19:12:57.283Z</timestamp>
        <server_timestamp>2014-02-23T19:12:59.848Z</server_timestamp>
        <history_uri>/ws/v1/streams/history/F9F967B4-33804DCE-2BDC4702-
45053B1C/temperature</history_uri>
    </stream>
</results>
```

Example: Create a stream

Use the following API to create (or update) one or more streams. The request payload can include a single stream or multiple streams. Multiple streams must be wrapped in an array for JSON and a <list> element for XML. The response always returns the streams as a list.

Request

```
POST /ws/v1/streams/inventory
```

```
{
    "description": "test stream",
    "id": "MyNewStream",
    "type": "LONG"
}
```

Response

```
{
    "count": 1,
    "list": [
        {
            "description": "test stream",
            "id": "MyNewStream",
            "type": "LONG"
        }
    ]
}
```

Example: Create multiple streams

The following example shows how to create multiple streams with one stream request.

Request

```
POST /ws/v1/streams/inventory
```

```
[
    {
        "description": "test stream",
        "id": "MyNewStream",
        "type": "LONG"
    },
    {
        "description": "another test stream",
        "id": "MyOtherStream",
        "type": "STRING"
    }
]
```

Response

```
{
    "count": 2,
    "list": [
        {
            "description": "test stream",
            "id": "MyNewStream",
            "type": "LONG"
        },
        {
            "description": "another test stream",
            "id": "MyOtherStream",
            "type": "STRING"
        }
    ]
}
```

Example: Add multiple data points to a data stream

The following example adds multiple data points to the "myStream" data stream.

Request

```
/ws/v1/streams/history
```

```
[{
  "stream_id": "myStream",
  "stream_type": "DOUBLE",
  "timestamp": "2014-02-23T19:37:04.517Z",
  "value": "41"
},
```

```
{
  "stream_id": "myStream",
  "stream_type": "DOUBLE",
  "timestamp": "2014-02-23T19:38:01.372Z",
  "value": "42"
},
{
  "stream_id": "myStream",
  "stream_type": "DOUBLE",
  "timestamp": "2014-02-23T19:39:02.785Z",
  "value": "43"
}]
```

Example: Edit a stream

Use the following API request to update (or create) a single stream.

Request

```
PUT /ws/v1/streams/inventory/F9F967B4-33804DCE-2BDC4702-45053B1C/temperature
```

```
{
    "units": "Celsius"
}
```

Response

```
{
    "description": "freezer",
    "history_uri": "/ws/v1/streams/history/F9F967B4-33804DCE-2BDC4702-
45053B1C/temperature",
    "id": "F9F967B4-33804DCE-2BDC4702-45053B1C/temperature",
    "server_timestamp": "2014-02-23T19:12:59.848Z",
    "timestamp": "2014-02-23T19:12:57.283Z",
    "type": "LONG",
    "units": "Celsius",
    "value": "71"
}
```

Example: Delete a stream

Use the following API request to get history for data points in a stream:

```
DELETE /ws/v1/streams/inventory/F9F967B4-33804DCE-2BDC4702-45053B1C/temperature
```

A DELETE request does not return a response unless there is an error.

Example: Get data history for a stream

Use the following API request to get history for data points in a stream:

Request

```
GET /ws/v1/streams/history/F9F967B4-33804DCE-2BDC4702-45053B1C/temperature
```

Response

```
{
    "count": 1000,
    "size": 1000,
    "cursor": "4fec418e-9cba-11e3-9a38-7cc3a1879642",
    "next_uri": "/ws/v1/streams/history/F9F967B4-33804DCE-2BDC4702-
45053B1C/temperature?cursor=4fec418e-9cba-11e3-9a38-7cc3a1879642",
    "list": [
        {
            "id": "1e1b67a5-9cb6-11e3-9a38-7cc3a1879642",
            "quality": 0,
            "server_timestamp": "2014-05-08T20:05:23.358Z",
            "stream_id": "F9F967B4-33804DCE-2BDC4702-45053B1C/temperature",
            "timestamp": "2014-02-23T18:12:55.434Z",
            "value": "68"
        },
        {
            "id": "4fec418e-9cba-11e3-9a38-7cc3a1879642",
            "quality": 0,
            "server_timestamp": "2014-05-08T20:06:09.710Z",
            "stream_id": "F9F967B4-33804DCE-2BDC4702-45053B1C/temperature",
            "timestamp": "2014-02-23T18:42:56.998Z",
            "value": "70"
        }
        ...
    ]
}
```

Example: Delete data points for a stream

Use the following API request to delete datapoints from a steam. You can delete datapoints within a time range. If no time range is specified, all datapoints are deleted.

```
DELETE /ws/v1/streams/history/F9F967B4-33804DCE-2BDC4702-
45053B1C/temperature?start_time=2014-02-23T00:00:00.000Z&end_time=2014-02-
24T00:00:00.000Z
```

Example_Get rollup data for a stream

Use the following API request to get rollups of stream history. Rollups are defined by a method (average, min, max, and so on) and an interval (hourly, daily, and so on).

Request

```
GET /ws/v1/streams/rollups/F9F967B4-33804DCE-2BDC4702-
45053B1C/temperature?method=average&interval=hour&start_time=2014-02-
23T00:00:00.000Z
```

Response

```
{
    "count": 2,
    "size": 1000,
    "start_time": "2014-02-23T00:00:00.000Z"
    "list": [
        {
            "stream_id": "F9F967B4-33804DCE-2BDC4702-45053B1C/temperature",
            "timestamp": "2014-02-23T18:00:00.000Z",
            "value": 69.0
        },
        {
            "stream_id": "F9F967B4-33804DCE-2BDC4702-45053B1C/temperature",
            "timestamp": "2014-02-23T19:00:00.000Z",
            "value": 71.0
        }
    ],
}
```

Example: Get carrier usage information

The following example shows how to get carrier usage information for your Remote Manager account. Usage information is reported by device. The id returned for each carrier subscription can be used as the stream_id on /ws/v1/streams requests.

Request

```
GET /ws/v1/streams/inventory?category=carrier
```

Response

```
{
  "count" : 1,
  "size" : 1000,
  "list" : [ {
    "id" : "0008CAFE-F4F71405-7CD7F7FF-
FFD126B5/carrier/89014103257651711449/usage/data/transferred",
    "type" : "DOUBLE",
    "value" : "0.0",
    "units" : "kbytes",
    "timestamp" : "2016-05-09T08:49:49.131Z",
```

```
    "server_timestamp" : "2016-05-09T08:49:49.173Z",
    "history_uri" : "/ws/v1/streams/history/0008CAFE-F4F71405-7CD7F7FF-
FFD126B5/carrier/89014103257651711449/usage/data/transferred"
  } ]
}
```

v1/users

Use the **users** web service to create, update, or list users for your account. Only admin users can view or change all users for an account. Users without admin privileges can view or update their own user account, but they cannot change their security policy or role.

URI

```
http://<hostname>/ws/v1/settings
```

Formats

HTTP method	Format	Description	Parameters
GET	/ws/v1/users	Get summary of the users APIs.	None
GET	/ws/v1/users/bulk	Get a list of all users for the account in CSV format.	orderby query
POST	/ws/v1/users/change_password/{uuid}	Change password for an existing user.	None
POST	/ws/v1/users/forgot_password/{username:.+}		None
POST	/ws/v1/users/forgot_username/{email_address:.+}		
GET POST	/ws/v1/users/inventory	Retrieve (GET) or change a list of all users in the account.	orderby cursor query size

HTTP method	Format	Description	Parameters
GET PUT DELETE	/ws/v1/users/inventory/{username:.+}	Retrieve (GET), create (PUT), or remove (DELETE) a user.	None

Fields

address

(Optional) String that specifies the street address for the user.

city

(Optional) String that specifies the city for the user.

country

(Optional) String that specifies the country for the user.

email

(Required) Email address for the user.

enabled

(Required) Specifies whether the username is enabled or disabled. The default is enabled.

first_name

(Required) First name of the user.

job_title

(Optional) Job title of the user.

last_login

Returns the timestamp of the last login for the user.

last_name

(Required) Last name of the user.

password

(Required) Password of the user.

phone_number

(Optional) Phone number for the user.

postal_code

(Optional) Postal code for the user address.

registration_date

Returns the timestamp for the user registration.

security_policy

(Optional) Specifies a security policy for the user.

role

(Required) Specifies the role for the user. Roles include: Administrator, User, Read only users, Application, Read only application.

state

(Optional) State for the user address.

username

(Required) Username for the user.

XbeeAttributeCore

Use the XbeeAttribueCore web service to identify one or more attributes of any node in your set of home area networks (HANs).

URI

```
http://<hostname>/ws/XbeeAttributeCore
```

Formats

HTTP method	Format	Description
GET	/ws/XbeeAttributeCore[?param1&param2...&paramn]	List all nodes in your account.

Elements

devConnectwareId

Device identifier of the node gateway.

xpExtAddr

ZigBee 64-bit extended address from the device.

xpParentAddr

For an endnode (xpNodeType = 2), the network address of the connecting router. For a router (xpNodeType = 1), the value is 0xFFFE.

xeEndpointId

ZigBee endpoint on which the cluster resides.

xpProfileId

ZigBee device profile associated with the node.

xeDeviceId

ZigBee device type associated with the node.

xeDeviceVersion

ZigBee device version.

xcClusterType

ZigBee cluster type.

Cluster type	Description
0	Server
1	Client

xaAttributeId

ZigBee attribute identifier.

xaAttributeType

ZigBee attribute type. See ZigBee Cluster Library (ZCL) and associated profile specification for more information on ZigBee attribute types.

Example: Identify node attributes in your home area networks (HANs)

The following example shows how to identify node attributes in your home area networks (HANs).

Request

```
/ws/XbeeAttributeCore
```

Response (abbreviated)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
   <resultTotalRows>5978</resultTotalRows>
   <requestedStartRow>0</requestedStartRow>
   <resultSize>1000</resultSize>
   <requestedSize>1000</requestedSize>
   <remainingSize>4978</remainingSize>
   <XbeeAttributeCore>
      <id>
         <xpExtAddr>00:08:A2:00:06:3D:8E:BC</xpExtAddr>
         <xeEndpointId>1</xeEndpointId>
         <xcClusterType>0</xcClusterType>
         <xcClusterId>1794</xcClusterId>
         <xaAttributeId>256</xaAttributeId>
      </id>
      <cstId>2</cstId>
      <devConnectwareId>00000000-00000000-000000FF-FF000B21</devConnectwareId>
      <xeProfileId>265</xeProfileId>
      <xeDeviceId>1281</xeDeviceId>
      <xeDeviceVersion>0</xeDeviceVersion>
      <xaAttributeType>37</xaAttributeType>
   </XbeeAttributeCore>
   <XbeeAttributeCore>
      <id>
         <xpExtAddr>00:08:A2:00:0D:F1:50:05</xpExtAddr>
         <xeEndpointId>1</xeEndpointId>
         <xcClusterType>0</xcClusterType>
         <xcClusterId>1794</xcClusterId>
         <xaAttributeId>256</xaAttributeId>
      </id>
      <cstId>2</cstId>
      <devConnectwareId>00000000-00000000-000000FF-FF000427</devConnectwareId>
      <xeProfileId>265</xeProfileId>
      <xeDeviceId>1281</xeDeviceId>
      <xeDeviceVersion>0</xeDeviceVersion>
      <xaAttributeType>37</xaAttributeType>
   </XbeeAttributeCore>
   <XbeeAttributeCore>
      <id>
         <xpExtAddr>00:08:A2:00:0E:C3:35:E2</xpExtAddr>
         <xeEndpointId>1</xeEndpointId>
         <xcClusterType>0</xcClusterType>
         <xcClusterId>1794</xcClusterId>
         <xaAttributeId>256</xaAttributeId>
      </id>
```

```
    <cstId>2</cstId>
    <devConnectwareId>00000000-00000000-000000FF-FF0004EF</devConnectwareId>
    <xeProfileId>265</xeProfileId>
    <xeDeviceId>1281</xeDeviceId>
    <xeDeviceVersion>0</xeDeviceVersion>
    <xaAttributeType>37</xaAttributeType>
</XbeeAttributeCore>
```

XbeeAttributeFull

Use the XbeeAttributeFull web service to display a list of ZigBee attribute names.

URI

```
http://<hostname>/ws/XbeeAttributeFull
```

Formats

HTTP method	Format	Description
GET	/ws/XbeeAttributeFull	List all ZigBee attributes for all nodes.

Elements

None

Example: List ZigBee full attrbutes

The following example shows how to get a complete list of ZigBee attributes.

Request

```
GET /ws/XbeeAttributeFull
```

Response (abbreviated)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
   <resultTotalRows>5978</resultTotalRows>
   <requestedStartRow>0</requestedStartRow>
   <resultSize>1000</resultSize>
   <requestedSize>1000</requestedSize>
   <remainingSize>4978</remainingSize>
   <XbeeAttributeFull>
      <id>
         <xpExtAddr>00:08:A2:00:06:3D:8E:BC</xpExtAddr>
         <xeEndpointId>1</xeEndpointId>
         <xcClusterType>0</xcClusterType>
         <xcClusterId>1794</xcClusterId>
         <xaAttributeId>256</xaAttributeId>
      </id>
      <cstId>2</cstId>
      <devConnectwareId>00000000-00000000-000000FF-FF000B21</devConnectwareId>
      <xpNetAddr>16054</xpNetAddr>
      <xpNodeType>1</xpNodeType>
      <xpMfgId>4126</xpMfgId>
      <xpDiscoveryIndex>1</xpDiscoveryIndex>
      <xpUpdateTime>2014-06-04T02:14:00.000Z</xpUpdateTime>
      <xeStatus>0</xeStatus>
      <xeProfileId>265</xeProfileId>
      <xeDeviceId>1281</xeDeviceId>
      <xeDeviceVersion>0</xeDeviceVersion>
      <xaAttributeType>37</xaAttributeType>
   </XbeeAttributeFull>
   <XbeeAttributeFull>
      <id>
         <xpExtAddr>00:08:A2:00:0D:F1:50:05</xpExtAddr>
         <xeEndpointId>1</xeEndpointId>
         <xcClusterType>0</xcClusterType>
         <xcClusterId>1794</xcClusterId>
         <xaAttributeId>256</xaAttributeId>
      </id>
      <cstId>2</cstId>
      <devConnectwareId>00000000-00000000-000000FF-FF000427</devConnectwareId>
      <xpNetAddr>16054</xpNetAddr>
      <xpNodeType>1</xpNodeType>
      <xpMfgId>4126</xpMfgId>
      <xpDiscoveryIndex>1</xpDiscoveryIndex>
      <xpUpdateTime>2014-06-04T02:01:00.000Z</xpUpdateTime>
      <xeStatus>0</xeStatus>
      <xeProfileId>265</xeProfileId>
      <xeDeviceId>1281</xeDeviceId>
```

```
      <xeDeviceVersion>0</xeDeviceVersion>
      <xaAttributeType>37</xaAttributeType>
   </XbeeAttributeFull>
   <XbeeAttributeFull>
      <id>
         <xpExtAddr>00:08:A2:00:0E:C3:35:E2</xpExtAddr>
         <xeEndpointId>1</xeEndpointId>
         <xcClusterType>0</xcClusterType>
         <xcClusterId>1794</xcClusterId>
         <xaAttributeId>256</xaAttributeId>
      </id>
      <cstId>2</cstId>
      <devConnectwareId>00000000-00000000-000000FF-FF0004EF</devConnectwareId>
      <xpNetAddr>16054</xpNetAddr>
      <xpNodeType>1</xpNodeType>
      <xpMfgId>4126</xpMfgId>
      <xpDiscoveryIndex>1</xpDiscoveryIndex>
      <xpUpdateTime>2014-06-04T02:01:00.000Z</xpUpdateTime>
      <xeStatus>0</xeStatus>
      <xeProfileId>265</xeProfileId>
      <xeDeviceId>1281</xeDeviceId>
      <xeDeviceVersion>0</xeDeviceVersion>
      <xaAttributeType>37</xaAttributeType>
   </XbeeAttributeFull>
   <XbeeAttributeFull>
      <id>
         <xpExtAddr>00:08:A2:00:21:46:E3:46</xpExtAddr>
         <xeEndpointId>1</xeEndpointId>
         <xcClusterType>0</xcClusterType>
         <xcClusterId>1794</xcClusterId>
         <xaAttributeId>256</xaAttributeId>
      </id>
      <cstId>2</cstId>
      <devConnectwareId>00000000-00000000-000000FF-FF00072E</devConnectwareId>
      <xpNetAddr>16054</xpNetAddr>
      <xpNodeType>1</xpNodeType>
      <xpMfgId>4126</xpMfgId>
      <xpDiscoveryIndex>1</xpDiscoveryIndex>
      <xpUpdateTime>2014-06-04T02:01:00.000Z</xpUpdateTime>
      <xeStatus>0</xeStatus>
      <xeProfileId>265</xeProfileId>
      <xeDeviceId>1281</xeDeviceId>
      <xeDeviceVersion>0</xeDeviceVersion>
      <xaAttributeType>37</xaAttributeType>
   </XbeeAttributeFull>
```

XbeeClusterCore

Use the XbeeClusterCore web resource to identify one or more clusters of any node in your set of home area networks (HANs). To retrieve XBee attributes or attribute data, use the XbeeAttributeCore or XbeeAttributeFull web services.

URI

```
/http://<hostname>/ws/XbeeClusterCore
```

Formats

HTTP method	Format	Description
GET	/ws/XbeeClusterCore[?param1&param2...&paramn]	List all clusters.

Elements

devConnectwareId

Device identifier of the node gateway.

xpExtAddr

ZigBee 64-bit extended address from the device.

xpParentAddr

For an endnode (xpNodeType = 2), the network address of the connecting router. For a router (xpNodeType = 1), the value is 0xFFFE.

xeEndpointId

ZigBee endpoint on which the cluster resides.

xpProfileId

ZigBee device profile associated with the node.

xeDeviceId

ZigBee device type associated with the node.

xeDeviceVersion

ZigBee device version.

xcClusterId

ZigBee cluster associated with the node.

xcClusterType

ZigBee cluster type.

Cluster type	Description
0	Server
1	Client

Example: List all clusters

The following example shows how to list all clusters for your account.

Request

```
/ws/XbeeClusterCore
```

Response (abbreviated)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
   <resultTotalRows>53802</resultTotalRows>
   <requestedStartRow>0</requestedStartRow>
   <resultSize>1000</resultSize>
   <requestedSize>1000</requestedSize>
   <remainingSize>52802</remainingSize>
   <XbeeClusterCore>
      <id>
         <xpExtAddr>00:08:A2:00:06:3D:8E:BC</xpExtAddr>
         <xeEndpointId>1</xeEndpointId>
         <xcClusterType>0</xcClusterType>
         <xcClusterId>0</xcClusterId>
      </id>
      <cstId>2</cstId>
      <devConnectwareId>00000000-00000000-000000FF-FF000B21</devConnectwareId>
      <xeProfileId>265</xeProfileId>
      <xeDeviceId>1281</xeDeviceId>
      <xeDeviceVersion>0</xeDeviceVersion>
   </XbeeClusterCore>
   <XbeeClusterCore>
      <id>
         <xpExtAddr>00:08:A2:00:06:3D:8E:BC</xpExtAddr>
         <xeEndpointId>1</xeEndpointId>
         <xcClusterType>0</xcClusterType>
         <xcClusterId>3</xcClusterId>
      </id>
      <cstId>2</cstId>
      <devConnectwareId>00000000-00000000-000000FF-FF000B21</devConnectwareId>
      <xeProfileId>265</xeProfileId>
      <xeDeviceId>1281</xeDeviceId>
      <xeDeviceVersion>0</xeDeviceVersion>
   </XbeeClusterCore>
   <XbeeClusterCore>
      <id>
         <xpExtAddr>00:08:A2:00:06:3D:8E:BC</xpExtAddr>
         <xeEndpointId>1</xeEndpointId>
         <xcClusterType>0</xcClusterType>
         <xcClusterId>1794</xcClusterId>
      </id>
      <cstId>2</cstId>
      <devConnectwareId>00000000-00000000-000000FF-FF000B21</devConnectwareId>
      <xeProfileId>265</xeProfileId>
      <xeDeviceId>1281</xeDeviceId>
      <xeDeviceVersion>0</xeDeviceVersion>
   </XbeeClusterCore>
```

XbeeCore

Use the XbeeCore web service to display a current list of ZigBee nodes in your account or refresh (discover) the node list. You can also use PUT to associate text with a specified node.

URI

```
http://<hostname>/ws/XbeeCore
```

Formats

HTTP method	Format	Description
GET	/ws/XbeeCore[?param1&param2...&paramn]	List all nodes in your account.
PUT	/ws/XbeeCore/{xpExtAddr}/{xpUserMetaData}	Add user-defined text metadata to a node.

Elements

cstId

Remote Manager identifier for the customer.

grpId

Remote Manager identifier for the customer group.

grpPath

Full path name of the specified group.

devConnectwareId

Device identifier of the node gateway.

xpExtAddr

ZigBee 64-bit extended address from the device.

xpNetAddr

ZigBee 16-bit network address of the node.

xpNodeType

ZigBee node type:

Node type	Description
0	Coordinator
1	Router
2	Endnode

xpParentAddr

For an endnode (xpNodeType = 2), the network address of the connecting router. For a router (xpNodeType = 1), the value is 0xFFFE.

xpProfileId

ZigBee device profile associated with the node.

xpMfgId

ZigBee manufacturing identifier of the node.

xpDeviceType

Device type of the node.

- **Product type:** Low order 16 bits.
- **Module type:** High order 16 bits.

Text descriptions for product and module types are returned by xmtModuleTypeDesc and xptProductTypeDesc.

xpNodeId

ZigBee node identifier.

xpDiscoveryIndex

Index within the list of nodes discovered on the home area network (HAN).

xmtModuleTypeDesc

Text description of the module type defined in xpDeviceType.

xptProductTypeDesc

Text description of the product type defined in xpDeviceType.

xpStatus

For Smart Energy nodes only. Connection status of the node: 0 for disconnected or 1 for connected.

xpUpdateTime

Time the node was last discovered.

xpUserMetaData

User-defined free-form text associated with a specified node.

Parameters

Use the following parameters to determine the data to be retrieved by a GET operation.

Parameter	Description
cache	Boolean value that indicates whether to use the cached node list. ■ **true:** Return the cached list of discovered nodes. ■ **false:** Return the current list of discovered nodes.
clear	Boolean value that specifies whether to clear the current node list. ■ **true:** Clear the current node list and discover/refresh the list. ■ **false:** Use the current node information.

Example: List all nodes

The following example shows how to list all nodes in all XBee networks associated with your Remote Manager account.

Request

```
GET /ws/XbeeCore
```

Response (abbreviated)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
   <resultTotalRows>5978</resultTotalRows>
   <requestedStartRow>0</requestedStartRow>
   <resultSize>1000</resultSize>
   <requestedSize>1000</requestedSize>
   <remainingSize>4978</remainingSize>
   <XbeeCore>
      <xpExtAddr>00:08:A2:00:06:3D:8E:BC</xpExtAddr>
      <devConnectwareId>00000000-00000000-000000FF-FF000B21</devConnectwareId>
      <cstId>2</cstId>
      <grpId>2</grpId>
      <xpNetAddr>16054</xpNetAddr>
      <xpNodeType>1</xpNodeType>
      <xpMfgId>4126</xpMfgId>
      <xpDiscoveryIndex>1</xpDiscoveryIndex>
      <xpStatus>1</xpStatus>
      <xpUpdateTime>2014-06-04T02:14:00.000Z</xpUpdateTime>
      <grpPath/>
   </XbeeCore>
   <XbeeCore>
      <xpExtAddr>00:08:A2:00:0D:F1:50:05</xpExtAddr>
      <devConnectwareId>00000000-00000000-000000FF-FF000427</devConnectwareId>
      <cstId>2</cstId>
      <grpId>2</grpId>
      <xpNetAddr>16054</xpNetAddr>
      <xpNodeType>1</xpNodeType>
      <xpMfgId>4126</xpMfgId>
      <xpDiscoveryIndex>1</xpDiscoveryIndex>
      <xpStatus>1</xpStatus>
      <xpUpdateTime>2014-06-04T02:01:00.000Z</xpUpdateTime>
      <grpPath/>
   </XbeeCore>
   <XbeeCore>
      <xpExtAddr>00:08:A2:00:0E:C3:35:E2</xpExtAddr>
      <devConnectwareId>00000000-00000000-000000FF-FF0004EF</devConnectwareId>
      <cstId>2</cstId>
      <grpId>2</grpId>
      <xpNetAddr>16054</xpNetAddr>
      <xpNodeType>1</xpNodeType>
      <xpMfgId>4126</xpMfgId>
      <xpDiscoveryIndex>1</xpDiscoveryIndex>
      <xpStatus>1</xpStatus>
      <xpUpdateTime>2014-06-04T02:01:00.000Z</xpUpdateTime>
      <grpPath/>
```

```
</XbeeCore>
<XbeeCore>
   <xpExtAddr>00:08:A2:00:21:46:E3:46</xpExtAddr>
   <devConnectwareId>00000000-00000000-000000FF-FF00072E</devConnectwareId>
   <cstId>2</cstId>
   <grpId>2</grpId>
   <xpNetAddr>16054</xpNetAddr>
   <xpNodeType>1</xpNodeType>
   <xpMfgId>4126</xpMfgId>
   <xpDiscoveryIndex>1</xpDiscoveryIndex>
   <xpStatus>1</xpStatus>
   <xpUpdateTime>2014-06-04T02:01:00.000Z</xpUpdateTime>
   <grpPath/>
</XbeeCore>
<XbeeCore>
   <xpExtAddr>00:08:A2:00:2D:DF:F3:25</xpExtAddr>
   <devConnectwareId>00000000-00000000-000000FF-FF000A16</devConnectwareId>
   <cstId>2</cstId>
   <grpId>2</grpId>
   <xpNetAddr>16054</xpNetAddr>
   <xpNodeType>1</xpNodeType>
   <xpMfgId>4126</xpMfgId>
   <xpDiscoveryIndex>1</xpDiscoveryIndex>
   <xpStatus>1</xpStatus>
   <xpUpdateTime>2014-06-04T02:14:00.000Z</xpUpdateTime>
   <grpPath/>
</XbeeCore>
<XbeeCore>
   <xpExtAddr>00:08:A2:00:40:CA:9D:2B</xpExtAddr>
   <devConnectwareId>00000000-00000000-000000FF-FF000673</devConnectwareId>
   <cstId>2</cstId>
   <grpId>2</grpId>
   <xpNetAddr>16054</xpNetAddr>
   <xpNodeType>1</xpNodeType>
   <xpMfgId>4126</xpMfgId>
   <xpDiscoveryIndex>1</xpDiscoveryIndex>
   <xpStatus>1</xpStatus>
   <xpUpdateTime>2014-06-04T02:15:00.000Z</xpUpdateTime>
   <grpPath/>
</XbeeCore>
<XbeeCore>
   <xpExtAddr>00:08:A2:00:40:F1:60:54</xpExtAddr>
   <devConnectwareId>00000000-00000000-000000FF-FF000B80</devConnectwareId>
   <cstId>2</cstId>
   <grpId>2</grpId>
   <xpNetAddr>16054</xpNetAddr>
   <xpNodeType>1</xpNodeType>
   <xpMfgId>4126</xpMfgId>
   <xpDiscoveryIndex>1</xpDiscoveryIndex>
   <xpStatus>1</xpStatus>
   <xpUpdateTime>2014-06-04T02:00:00.000Z</xpUpdateTime>
   <grpPath/>
</XbeeCore>
<XbeeCore>
   <xpExtAddr>00:08:A2:00:43:AE:22:86</xpExtAddr>
   <devConnectwareId>00000000-00000000-000000FF-FF0004FE</devConnectwareId>
   <cstId>2</cstId>
   <grpId>2</grpId>
   <xpNetAddr>16054</xpNetAddr>
```

```
      <xpNodeType>1</xpNodeType>
      <xpMfgId>4126</xpMfgId>
      <xpDiscoveryIndex>1</xpDiscoveryIndex>
      <xpStatus>1</xpStatus>
      <xpUpdateTime>2014-06-04T02:14:00.000Z</xpUpdateTime>
      <grpPath/>
   </XbeeCore>
```

Example: Request current list of nodes from a gateway

The following example shows how to request the current list of nodes from the gateway at address 00:13:A2:00:00:00:00:00.

```
GET /ws/XbeeCore?condition=xpExtAddr='00:13:A2:00:00:00:00:00'&cache=false
```

Example: Request node discovery

The following example shows how to request a fresh discovery of all nodes for gateway 00:13:A2:00:00:00:00:00.

```
GET /ws/XbeeCore?condition=xpExtAddr='00:13:A2:00:00:00:00:00'&clear=true
```

Example: Add a test label to a node

The following example shows how to add a text label to a specified node.

```
GET /ws/XbeeCore
```

```
<XbeeCore>
  <xpExtAddr>00:13:A2:00:00:00:00:00</xpExtAddr>
  <xpUserMetaData>user data here</xpUserMetaData>
</XbeeCore>
```

Deprecated APIs

The following APIs have been deprecated and should not be used in new code. For compatibility, deprecated APIs will be supported for a limited time, but code containing the deprecated APIs should be modified to use a supported API as soon as possible.

Resource path	Migrate to use
CarrierSubscription	`/ws/v1/streams/inventory?category=carrier`
CarrierUsage	`/ws/v1/streams/history/{device_id}/carrier/{sim_id}/usage/{usage_id}`
data	`ws/FileData`
DiaChannelDataFull	`ws/DataPoint/dia/channel/<Device Id>/<instance>/<channel>`
DiaChannelDataHistoryFull	`ws/DataPoint/dia/channel/<Device Id>/<instance>/<channel>`
XbeeAttributeDataCore	`ws/DataPoint/se/attr/<Device Id>/<XBee Address>/<Endpoint Id>/<Cluster Type>/<Cluster Id>/<Attribute Id>`
XbeeAttributeDataFull	`ws/DataPoint/se/attr/<Device Id>/<XBee Address>/<Endpoint Id>/<Cluster Type>/<Cluster Id>/<Attribute Id>`
XbeeAttributeDataHistoryCore	`ws/DataPoint/se/attr/<Device Id>/<XBee Address>/<Endpoint Id>/<Cluster Type>/<Cluster Id>/<Attribute Id>`
XbeeAttributeDataHistoryFull	`ws/DataPoint/se/attr/<Device Id>/<XBee Address>/<Endpoint Id>/<Cluster Type>/<Cluster Id>/<Attribute Id>`

XbeeAttributeReportingCore	Use the following SCI commands:
	`start_reports`
	`get_local_reporting_configurations`

XbeeEventDataCore	`ws/DataPoint/se/event/<Device Id>/<XBee Address>/<Endpoint Id>/<Cluster Type>/<Cluster Id>/<Attribute Id>`

Resource path	Migrate to use
XbeeEventDataFull	ws/DataPoint/se/event/<Device Id>/<XBee Address>/<Endpoint Id>/<Cluster Type>/<Cluster Id>/<Attribute Id>
XbeeEventDataHistoryCore	ws/DataPoint/se/event/<Device Id>/<instance>/<channel>
XbeeEventDataHistoryFull	ws/DataPoint/se/event/<Device Id>/<XBee Address>/<Endpoint Id>/<Cluster Type>/<Cluster Id>/<Attribute Id>

DIA (device integration application)

The Device Integration Application (DIA) is an application software platform for the Digi International family of gateway devices. Written in the Python programming language, DIA provides ready-to-use software that can be extended to meet custom device connectivity requirements. You can enable this feature using the DIA data service subscription.

Using DIA, you can build custom remote sampling solutions that report data through DIA. The reported data is stored in DIA data streams, and Remote Manager users can query and monitor the streams using the following format:

```
DataPoint/dia/channel/<Device ID>/<instance>/<channel>
```

SMS presentation data pushed to Remote Manager is also stored in DIA data streams.

Note This release supports storing the idigi_db presentation data for the DIA 1.3.8 or DIA 1.4 releases as shipped. If you customize the python source code for the idigi_db presentation, Remote Manager may not be able to parse the sample data uploaded from the device. If the idigi_db presentation cannot be parsed by Remote Manager, the sample data is stored in a file rather than DIA data streams.

Note Only the idigi_db presentation data is stored in the DIA data streams. RCI presentation data queried via Remote Manager is not stored in DIA data streams.

ISO 8601 date and duration reference

Date and Duration attributes must be expressed using ISO 8601 format. ISO 8601 is the International Standard for the representation of dates and times:

- ISO 8601 date format
- ISO 8601 duration format

ISO 8601 date format

Every component shown in the example below must be present when expressing a date in ISO 8601 format; this includes all punctuation characters and the "T" character. Within a string, the "T" indicates the beginning of the time element (directly following the date element). Although several date expressions exist, Remote Manager supports only the following format:

Complete date plus hours, minutes and seconds:

YYYY-MM-DDThh:mm:ss[.mmm]TZD (eg 2012-03-29T10:05:45-06:00)

Where:

YYYY = four-digit year
MM = two-digit month (eg 03=March)
DD = two-digit day of the month (01 through 31)
T = a set character indicating the start of the time element
hh = two digits of an hour (00 through 23, AM/PM not included)
mm = two digits of a minute (00 through 59)
ss = two digits of a second (00 through 59)
mmm = three digits of a millisecond (000 through 999)
TZD = time zone designator (Z or +hh:mm or -hh:mm), the + or - values indicate how far ahead or behind a time zone is from the UTC (Coordinated Universal Time) zone.

US time zone values are as follows:

EDT = -4:00
EST/CDT = -5:00
CST/MDT = -6:00
MST/PDT = -7:00
PST = -8:00

Note Use URL encoding (percent encoding)to include non-ASCII characters in an HTML request.

ISO 8601 duration format

ISO 8601 Durations are expressed using the following format, where (n) is replaced by the value for each of the date and time elements that follow the (n):

P(n)Y(n)M(n)DT(n)H(n)M(n)S

Where:

- **P** is the duration designator (referred to as "period"), and is always placed at the beginning of the duration.
- **Y** is the year designator that follows the value for the number of years.
- **M** is the month designator that follows the value for the number of months.
- **W** is the week designator that follows the value for the number of weeks.
- **D** is the day designator that follows the value for the number of days.
- **T** is the time designator that precedes the time components.
- **H** is the hour designator that follows the value for the number of hours.
- **M** is the minute designator that follows the value for the number of minutes.
- **S** is the second designator that follows the value for the number of seconds.

For example:

P3Y6M4DT12H30M5S

Represents a duration of three years, six months, four days, twelve hours, thirty minutes, and five seconds.

HTTP interface specification

All HTTP operations require basic authentication. The HTTP basic authentication user is set to the device ID and the password is set to the device password.

A device ID and password can be generated by Remote Manager by performing a POST to /ws/DeviceCore and specifying the <provisionId> and <dpCurrentConnectPw> values. See DeviceCore.

Create a device ID

The following example shows the steps for creating a device ID used in a python program to upload a data file to Remote Manager.

To create a device ID (if you haven't already):

1. Log into Remote Manager.
2. Click **Admin > Account Settings > My Account**.
3. Within the vendor information section, copy your vendor ID number if one exists. If one does not exist, click **Register Vendor** or use the ws/DeviceVendor web service to create or retrieve a vendor ID. See DeviceVendor for more information.

   Note This example uses the vendor ID 0x0100001D. Replace the vendor ID with your own vendor ID.

4. Click **Documentation > API Explorer** to use the explorer to generate a device ID.
5. Perform a POST on /ws/DeviceCore using the following specifics:
   - Path: /ws/DeviceCore
   - HTTP Method: POST
   - Paste the following into the text area:

```
<DeviceCore>
   <provisionId>0x0100001D</provisionId>
   <dpCurrentConnectPw>DevicePassword123</dpCurrentConnectPw>
   <dpRestrictedStatus>0</dpRestrictedStatus>
   <grpPath />
   <dpDescription>My Simple Python HTTP Data Upload Program</dpDescription>
</DeviceCore>
```

   - <provisionId>: replace the Vendor ID (0x0100001D shown above) with your Vendor ID
   - <dpCurrentConnectPw>: select a device password for your device to use
   - <dpRestrictedStatus>: 0 means the device is allowed to send messages to Remote Manager
   - <grpPath/>: instructs Remote Manager to provision the new device ID in your root group
   - <dpDescription>: optional, but can only be set at creation time, so it's a good idea to supply one
   - <dpContact> and <dpLocation> can also be set at this time
6. Within the API Explorer, click **Send**to execute the operation. Remote Manager responds with a 201 (displayed in the Web Services Responses window)
7. Open the response.
8. The <result> tag contains the ID of the newly created device ID.
9. Copy the contents of the <result><location> tag (for example: DeviceCore/12345/0) and paste them into the Path field.
10. Select GET and then click the **Send** button.
11. Open the result and find the devConnectwareID; this is your new device ID.

12. Copy and paste the Python program shown in Python sample program into an editor, filling in the device ID and device password used above.
13. Run the program.

Uploading data to Remote Manager

```
PUT /ws/Messaging/<path>
```

Body: Remote Manager message contents

Header:

- Content-Type: (optional)
- delete: (optional) See Delete below.
- deleted: (optional) See Delete below.

Parameters:

A client sends Remote Manager a message in the form of a file located at `<path>`. `<path>`. This can be a simple file name, or a directory path. Remote Manager places the file in the specified path within the device's Data Service which can be accessed via Remote Manager or through Web Services using the /ws/FileData interface. The file name and the contents of the file are user defined.

The HTTP upload can specify a content type, for example (Content-Type: text/xml). If no content type is specified in the header, the content type is implied by the file name extension (example: filename.xml).

Returns:

If the upload is successful, the server sends an HTTP 200 status code, and it will also provide back a list of messages that are waiting for the device in the payload. The format of the response is the same as that returned by doing a GET to /ws/Messaging. See Retrieve files ready for the device and Retrieve a specific message for a device for information on the structure of the information returned by performing a GET to /ws/Messaging.

Data limits related to direct device uploads

The Simple HTTP Interface has the following limits:

- Number of data points allowed per request: 250
- Individual data point - maximum total size: 64KB
- Maximum upload size: 2MB

Sending a message to a device

The Simple HTTP Device Interface supports a simple mechanism that allows users to retrieve messages from Remote Manager.

A folder for the device called **inbox** is created in the Remote Manager Data Services folders list. Devices can retrieve a list of the contents of this folder, GET a message, and DELETE messages.

```
GET /ws/Messaging[/message1]
```

See Deleting a message from a device inbox

Retrieve files ready for the device

A GET to /ws/Messaging (without specifying a file name) returns a comma delimited list of messages waiting for the device in that device's inbox:

```
message1 name, message1 size, message 1 timestamp (milliseconds since epoch) …
message2 name, message2 size, message 2 timestamp (milliseconds since epoch)…
…
```

Retrieve a specific message for a device

A device can retrieve a specific message via a GET to /ws/Messaging/[messagename]. For example, a GET to /ws/Messaging/message1 returns the contents of message1.

The device will be returned the exact contents of the payload with no wrapper. If the requested message does not exist, the server will return a HTTP 404 message saying it does not exist.

Deleting a message from a device inbox

After a message has been processed by a device, the device may delete that message from the inbox. This can be used as a way of confirming the delivery of the message to the device.

There are two ways to delete inbox messages:

On any GET or PUT request, an optional delete: header can be specified with a comma delimited list of messages to delete. The response to these requests will contain a deleted: header with a list of deleted messages.

```
DELETE /ws/Messaging/<message>
```

Deletes <message> from the device's inbox.

Additionally, a list of messages ready for the device are included in the response payload. See section D.2.3 for information on the format.

Example: Post sensor readings using Python

The program uploads 10 sample readings. The results can be viewed in Remote Manager within the Data Services, Data Streams view page. To open this page click the Data Services tab, then select the Data Streams menu.

```
#############################################################
import httplib
import base64
import time
import random
"""
This program demonstrates the Simple HTTP Device Interface which
allows any client to upload data using only simple HTTP operations.
To use this program, first create a Device ID by:
POSTing to /ws/DeviceCore
<DeviceCore>
  <provisionId>...insert vendor id here...</provisionId>
  <dpCurrentConnectPw>...insert device password here...</dpCurrentConnectPw>
  <dpRestrictedStatus>0</dpRestrictedStatus>
  <grpPath/>
</DeviceCore>
```

```
This will create a new unique Device ID with the specified password and
adds it to your account.
Fill in the info below and then execute the program.
This will upload 10 samples.  The results can be viewed in Remote Manager by
navigating to the Data Services page and selecting Time Series data.
You can also view the results by using the
/ws/DiaChannelDataHistoryFull web service.
"""
# fill these values in with your settings:
deviceCloudName = "https://remotemanager.digi.com"
deviceId = "00080002-00000000-02000434-C69F5998"
devicePwd = "DevicePasswordGoesHere"
fileName = "tempSample.xml"
######################################################

def sendReading(reading):
    statuscode = -1
    statusmessage = "Request not sent"
    header = "none"
    response_body = "none"
    try:
        try:
            deviceCloud = deviceCloudName
            username = deviceId
            password = devicePwd
            filename = fileName
            body = reading
            # create HTTP basic authentication string, this consists of
            # "username:password" base64 encoded
            auth = base64.encodestring("%s:%s" % (username,password))[:-1]
            # Note, this is using Secure HTTP
            webservice = httplib.HTTPS(deviceCloud)
            # to what URL to send the request with a given HTTP method
            webservice.putrequest("PUT", "/ws/Messaging/%s" % (filename))
            # add the authorization string into the HTTP header
            webservice.putheader("Authorization", "Basic %s" % (auth))
            webservice.putheader("Content-type", "text/xml; charset=\"UTF-8\"")
            webservice.putheader("Content-length", "%d" % len(body))
            webservice.endheaders()
            webservice.send(body)
            # get the response
            statuscode, statusmessage, header = webservice.getreply()
            response_body = webservice.getfile().read()
        except Exception as e:
            print "PUT of data reading threw an exception: %s" % e
    finally:
        pass
    if statuscode == 200 or statuscode == 201:
        # it worked.  We're done
        return True
    else:
        print "************************************************************"
        print "PUT to /ws/Messaging failed.  Details:"
        print "statuscode: %d" % (statuscode)
        print "statusmessage: %s" % (statusmessage)
        print "header: %s" % (header)
        print "response: %s" % (response_body)
        print "************************************************************"
        return False
```

```
if __name__ == "__main__":
    sampleStart = "<idigi_data compact=\"True\">"
    sampleTemplate = "<sample name=\"%s\" value=\"%s\" unit=\"%s\" />"
    sampleEnd = "</idigi_data>"
    # This name can be changed to any name of the form: group.field
    name = "pythonProgram.temp"
    initialValue = 33.5
    units = "F"
    # this generates some fake data for interesting viewing
    for i in xrange (0, 10) :
        value = initialValue + i + (random.random() - 0.5) * i
        sample1 = sampleStart
        sample1 += sampleTemplate % (name, value, units)
        sample1 += sampleEnd
        print sample1
        sendReading(sample1)
        time.sleep(2)
```

UI descriptor reference

Menu templates

Under the root (ui) element a navigation element contains the menu template. Each menu is composed of a unique ID, name (display text), and associated page template name. The page and data groups that the page handles are optional and are typically not supplied if the menu has sub-menus. If the data groups the page handles are not listed and it has a page that is user defined it will parse the page contents and generate the field itself.

Example:

```
<ui>
   <navigation>
      <menu id="test1" name="Test" page="test_page"
data="settings:mgmtconnection/*" />
      <menu id="test2" name="Test page 2" required="true">
         <menu id="test2child" name="Test Child" page="default_properties_page"
dataRootDefault="settings" required="false" data="doesNotExist/*/desc" />
      </menu>
      <menu id="advanced_cfg" name="Advanced Configuration" page=""
dataRootDefault="settings" data="" required="false" organizeByGroup="true"
readonly="false" indexBy="">
         <automenu page="" dataRootDefault="settings" data="settings:*"
readonly="false" />
      </menu>
   </navigation>
</ui>
```

Menu element

The menu element represents a menu item. The menu hierarchy will be generated in the same parent/child relationship as XML menu elements recursively.

For example:

```
<ui>
   <navigation>
      <menu id="example_menu1" name="Single root level item" required="true" />
      <menu id="example_menu2" name="Parent root level item" required="true">
         <menu id="example_menu3" name="Child menu" required="true">
            <menu id="example_menu3" name="Sub-Child menu" required="true" />
         </menu>
      </menu>
   </navigation>
</ui>
```

Rendors as:

id (required)

The id attribute is required and must be unique. This is used to reference this menu item.

data

The data reference for a menu determines what property groups the associated page is responsible for. Property groups are, in RCI terms, settings or state groups. They are specified in the menu as a comma separated list of groups. Each group name can have an index (or dictionary name) specified in a slashed notation. For example: 'serial/1' OR 'tcp_echo,udp_echo,http,https'. A special data value of '*' is used to automatically generate menus for all property groups not already specified explicitly in the other menu items. This should be the last menu item specified and is typically placed under an 'Advanced' parent menu item.

name (required)

The name attribute is the label for the menu. It will be displayed in the menu and at the header of the property page when the menu is selected.

page

The page attribute is optional and used to specify what to render in the properties page when the menu item is selected. This may be either a Remote Manager provided page, like file management, or a custom page defined later in this document. If this is left blank then the property page will either be blank itself or will list any children menu items. If you want a page that lists all settings designated by the "data" attribute set this value to "default_properties_page" which is a pre-defined Remote Manager properties page. This is the equivalent of creating a page with the contents being just an `<unprocessed/>` element (see the Page Contents section for more information).

required

Boolean defining if this menu should be displayed even if the data listed in the data attribute does not exist. If this is set to false (default) and the query_settings of the device does not contain any of the properties listed in the data this menu will be removed.

dataRootDefault

Default root for the data fields when not explicitly specified (optional, settings is default).

organizeByGroup

Determines if page information is organized by group or in the order specified in data.

indexBy

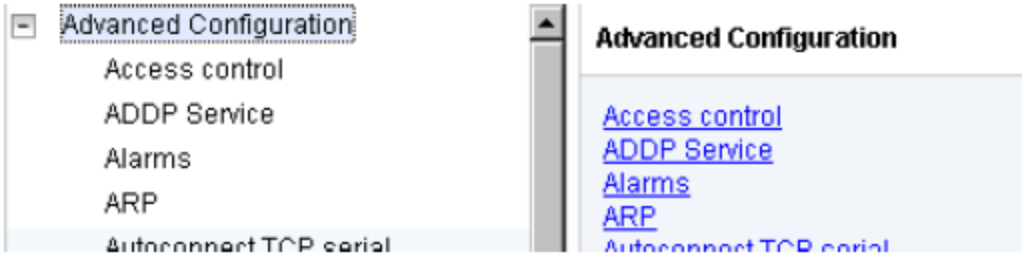
Default root for the data fields when not explicitly specified (optional, settings is default).

The automenu will render all settings for a device that has not been reserved by a different menu item via the data attribute.

Automenu

The automenu renders all settings for a device that have not been reserved by a different menu item via the data attribute.

Example



id

An optional unique identifier for this menu.

dataRootDefault

Default root for the data fields when not explicitly specified (optional, settings is default).

data

Comma separated list of the page's data fields (optional).

readonly

If all pages should render read-only.

Page templates

HTML templates.

Example:

```
<ui>
   <content>
      <page id="test_page" help="test_page_help">
         <b>My IP setting:</b>
         <property rciId="settings:mgmtconnection/1/serverAddress" />
```

```
         <hr />
         <h1>Advanced:</h1>
         <unprocessed>
            <exclude rciId="settings:mgmtconnection/1/timedConnectionPeriod" />
         </unprocessed>
      </page>
   </content>
</ui>
```

id

The id attribute is required and must be unique. This is used as a reference in the menus.

help

A reference to the id attribute of a help element shared within the content parent.

Page contents

The page contains xhtml and allows the following tags: b, p, i, s, a, img, table, thead, tbody, tfoot, tr, th, td, dd, dl, dt, em, h1, h2, h3, h4, h5, h6, li, ul, ol, span, div, strike, strong, sub, sup, pre, del, code, blockquote, strike, br, hr, small, big, property, unprocessed, exclude. Most of these tags are standard HTML and will be rendered accordingly in the page area of the device properties when its corresponding menu is selected.

```
<page id="test_page">
   <b>My Remote Manager server:</b>
   <property rciId="settings:mgmtconnection/1/serverAddress" />
   <hr />
   <h1>Advanced:</h1>
   <unprocessed>
      <exclude rciId="settings:mgmtconnection/1/timedConnectionPeriod" />
   </unprocessed>
</page>
```

Unprocessed tag

All data that is reserved by the menu pointing to this page that has not already been displayed by a property tag will be listed. There is an optional 'exclude' element as a child which will remove specific settings from this list.

```
<page id="test_page">
   <h1>Advanced:</h1>
   <unprocessed>
      <exclude rciId="settings:mgmtconnection/1/timedConnectionPeriod" />
   </unprocessed>
</page>
```

Help templates

HTML templates

```
<ui>
   <content>
```

```
      <help id="test_page_help">
         <b>Help</b>
         <h1 style="color:red">lots of HTML options!</h1>
      </help>
   </content>
</ui>
```

Help templates contain an id attribute that are used as reference. The contents are xhtml which will be displayed in a pop-up when the help option is clicked within the properties page.