



Device Cloud Programming Guide

Device Cloud Programming Guide

©2013 Digi International Inc.
All Rights Reserved.

Digi, Digi International, the Digi logo, the Digi website, Etherios, the Etherios logo, the Etherios website, Device Cloud by Etherios, Etherios Cloud Connector, XBee, and ConnectPort are trademarks or registered trademarks of Digi International, Inc. in the United States and other countries worldwide.

All other trademarks mentioned in this document are the property of their respective owners. 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, either 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.

This product could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes may be incorporated in new editions of the publication.

1. Introduction

1.1 Overview

Device Cloud by Etherios™ is a machine-to-machine cloud-based network operating platform that includes a variety of Application Programming Interfaces (API's). Device Cloud supports application to device data interaction (messaging), application & device data storage, and 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 pass data (messages) as well as send data to a temporary data cache on Device Cloud to be available for retrieval by Web Services clients using Device Cloud devices, DIA (Device Integration Application), or connecting your own device using Etherios® Cloud Connector™.

1.2 Connecting Applications to Device Cloud

Device Cloud provides a standard HTTP API that allows many ways to access data. Files and information about files can be accessed by:

- A standard browser by typing in the appropriate URL
- A Google Gadget
- A Java Application
- A Python Application running on a PC
- 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, etc.

2. Concepts

2.1 Subscriptions

Subscriptions in Device Cloud control what features are available on a customer by customer basis. When you sign up for an initial free Device Cloud account, you are automatically subscribed to the majority of available Device Cloud services. Subscriptions are used to turn Device Cloud services on and off.

Access to Web Services for an account is controlled via subscriptions. If you do not have a required subscription to access a Web Service, you will get a 403 return code.

2.2 Device IDs

Device IDs are used to identify devices in Device Cloud. A Device ID is a 16-octet number that is unique to the device and does not change over its lifetime. A Device ID is derived from globally unique values already assigned to a device (such as a MAC address, IMEI, etc). In resource web services, Device IDs are listed as devConnectwareId elements. The canonical method for writing Device IDs is as four groups of eight hexadecimal digits separated by a dash. An example Device ID is:

01234567-89ABCDEF-01234567-89ABCDEF

Device IDs may also be written in an abbreviated form. In this form, any leading groups that are all zeros may be omitted. At least one group must be specified.

Full Device ID	Abbreviated Forms
00000000-89ABCDEF-01234567-89ABCDEF	89ABCDEF-01234567-89ABCDEF
00000000-00000000-01234567-89ABCDEF	00000000-01234567-89ABCDEF 01234567-89ABCDEF
01234567-89ABCDEF-01234567-89ABCDEF	No abbreviated form; use full form
00000000-00000000-00000000-89ABCDEF	00000000-00000000-89ABCDEF 00000000-89ABCDEF 89ABCDEF

2.2.1 Device ID Assignment Convention

Generally a Device ID is generated from the unique information from the list below, in the order specified. 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 2 Ethernet interfaces), a "primary" interface is selected and used as the source of the Device ID.

1. Ethernet interface MAC-48
2. 802.11 interface MAC-48
3. Use the cellular modem IMEI if GSM
4. Use the cellular modem ESN if CDMA
5. Use the Orbcomm global unique id

2.2.1.1 48-bit MAC Address Format

- 12 hex digits
- mapping to CWID
- top 64 bits set to zero
- lower 64 bits using EUI-64 format (MAC-48 extension). 48-bit MAC is a special case of EUI-64. The mapping from EUI-48 to EUI-64 is a standard mapping specified in EUI-64.

Example MAC: 112233:445566

Device ID mapping: 00000000-00000000-112233FF-FF445566

2.2.1.2 GSM IMEI

- 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, it is included in the Device ID mapping.

Example IMEI: AA-BBBBBB-CCCCC-D

Device ID mapping: 00010000-00000000-0AABBBBB-BCCCCCDD

2.2.1.3 CDMA ESN/MEID

CDMA has two addressing schemes. An older ESN scheme which was a 32 bit address and MEID which is a 56 bit scheme which translates into 14 hex digits. Both addresses can be specified in either hex or decimal format.

Similar to IMEI a check digit is appended to MEID addresses but is not considered part of the MEID. It is included in the Device ID mapping.

MEID is actually compatible with IMEI since the first two digits of an MEID will always be $\geq 0xA0$ while those digits in an IMEI will always be less than $0xA0$. However, IMEI and MEID will have separate specifications for the Device ID.

Example ESN- Hex: MM-SSSSSS

Device ID mapping: 00020000-00000000-00000000-MM-SSSSSS

Example MEID- Hex: RR-XXXXXX-ZZZZZZ-C

Device ID mapping: 00040000-00000000-0RRXXXXX-XZZZZZZC

The decimal forms of these addresses are longer, with the Decimal MEID extending into the third word of the Device ID as shown in the table above.

2.2.1.4 Orbcomm Addresses

Orbcomm globally unique addresses can be used to generate a Device ID.

Orbcomm GID are 14 decimal digits with an 'M' in front. A Device ID is generated by stripping the M:

Orbcomm GID: M12345678901234

Device ID: 00070000-00000000-00123456-78901234

2.3 Embedded Device Development

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

When a device connects to Device Cloud, it supplies a Vendor ID and device type. The Vendor ID is the namespace where the particular vendor's device types exist. A device manufacturer using Vendor ID 3000 could create a device of type "iVendingMachine", and it would perfectly coexist with a device of type "iVendingMachine" by Vendor ID 3001. For more information on using third party devices with Device Cloud, see the Etherios Cloud Connector Getting Started Guide and Etherios Cloud Connector User Guide. For more information about Cloud Connector see www.etherios.com/devicecloud/connector.

The following process is used to create a Device Cloud enabled device.

1. Install a development kit.
2. Create an account on login.etherios.com.
3. Within Device Manager, navigate to the My Account page. To locate this page, select the My Account option from the Active User drop-down menu (in the upper right corner of the page), or select the My Account tab within the Account Settings page of the Admin tab.
4. Click on the Register for new vendor id button to generate a Vendor ID.
5. Put the Vendor ID in the application for the device and configure the device type to something meaningful and specific for the device.
6. Build the application and deploy it to the device.
7. When the application starts it will connect to Device Cloud. Device Cloud sees it is a new device type for that Vendor ID and queries the device for its descriptors.
8. Select the Devices menu within the Device Manager tab to open the Devices page and then locate your device within the Device list.
9. Double-click on your device to open the device's Properties page to see the descriptors rendered the settings/state of the device.
10. Return to the Devices page and click the More button within the toolbar and then select the Edit Descriptors option from the UI Descriptors category of the More Menu. Create a view descriptor then re-open the device's Properties page and see the settings/state rendered with additional format provided by the view descriptor.
11. Edit the application to have custom settings/state exposed. Build and deploy the application to the device.
12. Within the Devices page, click the More button within the toolbar and then select the Refresh Descriptors option from the UI Descriptors category of the More Menu to retrieve the latest descriptors.
13. Navigate to the device's Properties page and see custom settings/state exposed in the UI.

2.4 Data Services

Device Cloud's data services facilitates data collection from remote devices. Devices can push data files up to the server that are cached temporarily in a database on Device Cloud. Files are kept in collections, which are similar to folders. Client applications can get the data from these collections and then do something with it, such as displaying it in a graph on a web page. Files and collections can be accessed through the user portal view of Device Cloud and via Web Services.

2.4.1 Path Information

The path to a database collection or file is specified using a relative path. A user's home collection can be represented by a ~. As an example, to access the mydata collection under the user's home collection they could specify a URL like this:

~/mydata

A collection for a device includes the Device ID.

~/00000000-00000000-00000000-12345678

2.5 Device Information Caching

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

By default, the cache is automatically used to satisfy requests for information however the caller does have some ability to control this. When issuing an SC1 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 essentially 'refresh' the contents of the cache. The caller can also specify cache=only to instruct the server to only provide responses from the cache 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 don't want to incur the overhead of communicating with the device.

The user can control the amount of data that is actually returned from their request. Sometimes the users are simply sending a message to their devices and they don't really care about the responses they may get from those commands. By using the reply attribute of the send_message command, users can specify reply=all to get all reply data, reply=error to get only error replies, or reply=none to get none of the replies. Although the user can specify how much of the reply data is streamed back to them, Device Cloud servers will still inspect the reply data and update its various data caches appropriately.

2.5.1 Device Meta Data Cache

This cache stores information that is pertinent to all devices of a specific type. This information could be a descriptor that specifies all the settings or state of a device. It could be a copy of the default setting values of a device. These pieces of information are considered meta data for that device type and each one is stored under a unique name so as to differentiate it with other information for that same device. Identifying the type of device can be a bit tricky as it involves looking at a number of indicators on the device. The indicators used are:

- **DeviceType:** defines the hardware platform of the device (i.e. ConnectPort X2)
- **VendorID:** defines the vendor configured as owner for this type of device
- **ProductID:** defines the part number that Digi has assigned to this product
- **FirmwareId:** defines the kind of firmware loaded on this device (i.e. Smart Energy variant)
- **FirmwareLevel:** defines the revision number of the software (i.e. 2.9.1.0)

Together these five elements uniquely identify the 'kind' of device we are working with. Combining this with the name of the element being stored (i.e. descriptor/setting) identifies a piece of meta-data.

SCI Interface: The following RCI commands deal with the DeviceMetaData cache:

```
<query_descriptor><query_setting></query_descriptor>
```

This command fetches the requested descriptor for the device and returns it to the caller. In this case we are requesting a descriptor for the `<query_setting>` command, however, you can fetch descriptors for other commands as well. If the descriptor is available in the cache it will be returned immediately to the caller. If it is not available, the request will be forwarded to the device to see if it can supply the descriptor. If it can, the server will cache the response for future requests. If the device cannot supply its descriptor, then the server will go through some additional logic to locate a descriptor that is 'close' to what was requested. The logic to find a closest matching descriptor is as follows:

1. Look for matching device type with an older level of firmware
2. Look for matching device type with current or older level of firmware and a default (blank) ProductId
3. Look for matching device type with current or older level of firmware and a default (blank) ProductId & FirmwareId
4. Look for matching VendorID with current or older level of firmware and a default (blank) DeviceType, ProductId, & FirmwareId
5. Look for any current or older level of firmware and a default (blank) DeviceType, VendorID, ProductId, & FirmwareId

If we go through all this work and still can't find a descriptor then we return an unknown command error to the caller.

```
<query_setting source="internal_defaults">
```

This command fetches the internal default settings of the device and returns them to the caller. Similar to the descriptor, if the default settings are found in the cache, they are returned immediately to the caller, otherwise the device is queried for its defaults. The response of the device is stored in the cache for future requests and is then returned to the caller.

NOTE: Older devices will respond with current settings when queried for this data. The only way to tell if the device is responding with internal_defaults or current settings is to inspect the result element for the source="internal_defaults" attribute. Older firmware will not report this attribute.

NOTE: There is no fallback logic when default settings cannot be located.

```
<query_setting/>
```

This command primarily deals with the DeviceData cache since it is a request for device specific data. However, the implementation of the server uses both the DeviceMetaData cache and the DeviceData cache to satisfy the request. This is described in more detail later in the section on delta settings processing.

2.5.2 Device Data Cache

This cache stores information pertinent to a specific device instance. This information is generally settings and state data. The information is uniquely identified by the DeviceID and a unique name (i.e. state).

SCI Interface: The following RCI commands deal with the DeviceData cache:

```
<query_setting>
```

This command fetches the current configuration settings of the device and returns them to the caller. If the data is found in the cache it is returned directly, otherwise the request is sent to the device and the results are stored in the cache for future use before returning them to the user. This command also makes use of the DeviceMetaData cache as described in the section on delta settings processing.

```
<query_state>
```

This command fetches the current runtime state of the device and returns it to the caller. This command is handled almost identically to the query_setting command.

2.5.3 XBee Node Cache

This cache stores a list of known remote ZigBee network nodes. The information stored in this cache includes some general information about each node in the mesh as well as what gateway it is associated with.

SCI Interface: The following RCI commands deal with the XBee Node cache:

```
<do_command target="zigbee"><discover></do_command>
```

This command discovers all mesh nodes associated with the specified device and returns them to the caller. If the data is found in the cache it is returned directly. If it is not found, or the caller specifies `cache=false` then the request is sent to the device and the results are used to update the cache and returned to the caller. The discover takes an optional attribute `option="clear"` which informs the device to forget all devices it formerly knew and rediscover from scratch.

Resource Interface: The following web service URL can be used to view this cache: /ws/XbeeCore

2.5.4 XBee Data Cache

This cache stores information pertinent to a specific XBee® device instance. Like the DeviceData cache this generally means the configuration settings and state of the XBee node. The information is uniquely identified by the XBee radio's extended address and a unique name such as setting or state.

SCI Interface: The following RCI commands deal with the XBeeData cache:

```
<do_command target="zigbee"><query_setting addr="00:13:a2:00:40:0a:26:2c!"/></do_command>
```

This command fetches the settings of the specified ZigBee node and returns them to the caller. If the data is found in the cache it is returned directly. If it is not found, or the caller specifies `cache=false` then the request is sent to the device and subsequently to the XBee node and the results are used to update the cache and returned to the caller.

```
<do_command target="zigbee"><query_state addr="00:13:a2:00:40:0a:26:2c!"/></do_command>
```

This command fetches the state of the specified ZigBee node and returns them to the caller. If the data is found in the cache it is returned directly. If it is not found, or the caller specifies `cache=false` then the request is sent to the device and subsequently to the XBee node and the results are used to update the cache and returned to the caller.

3. Writing Web Services Client Applications

Device Cloud provides a REST-style API over HTTP (or HTTPS). Users can write HTTP clients in their preferred programming language that get data from Device Cloud and use or display the data in the way that they desire. Examples of such clients include Web pages and programs written in a language such as Python or Java. These clients send requests to the server using standard HTTP requests. The HTTP requests that Device Cloud supports are GET, PUT, POST, and DELETE. The server supports basic HTTP authentication and only valid users can access the database. To reduce the authentication overhead of multiple requests, either use an HTTP library that caches cookies, or cache the cookies JSESSIONID and SID yourself.

3.1 In a Web Browser

Any GET request can be typed into the URL field of a web browser. Some browser plug-ins also allow other HTTP methods to be called.

3.2 In Device Manager

Device Manager has an API Explorer that can be used to run any web service request.

3.3 Python

Python scripts can be written to send standard HTTP requests to the server. These scripts use Python libraries to handle connecting to the server, sending the request, and getting the reply. Below is a code snippet of an HTTP POST of an SCI request written in Python.

Device Manager has an API Explorer that can be used to run any web service request and export it as a Python application.

```
# The following lines require manual changes
username = "YourUsername" # enter your username
password = "YourPassword" # enter your password
device_id = "Target Device Id" # enter device id of target
# Nothing below this line should need to be changed
# -----
import httplib
import base64

# create HTTP basic authentication string, this consists of
# "username:password" base64 encoded
auth = base64.encodestring("%s:%s" % (username, password))[:-1]

# message to send to server
message = """<sci_request version="1.0">
<send_message>
  <targets>
    <device id="%s"/>
  </targets>
  <rci_request version="1.1">
    <query_state/>
  </rci_request>
</send_message>
</sci_request>
"""%(device_id)

webservice = httplib.HTTP("login.etherios.com ", 80)

# to what URL to send the request with a given HTTP method
webservice.putrequest("POST", "/ws/sci")

# 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(message))
webservice.endheaders()
webservice.send(message)

# get the response
statuscode, statusmessage, header = webservice.getreply()
response_body = webservice.getfile().read()

# print the output to standard out
print (statuscode, statusmessage)
print response_body
```

3.4 Java

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

Device Manager has an API Explorer that can be used to run any web service request and export it as a Java application.

```
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 Device Cloud web service.
 */
public class WebServiceRequest {

    /**
     * Run the web service request
     */
    public static void main(String[] args) {
        try {
            // Create url to Device Cloud server for a given web service request
            URL url = new URL("http://login.etherios.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("  </sci_request version='1.1'> \r\n");
out.write("    <query_state> \r\n");
out.write("  </sci_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();
}
}
}

```

4. Resource Conventions

4.1 URL Specification

Device Cloud Web Services APIs are RESTful in nature. Every URL relates to a specific resource or list of resources.

For example, the following URL retrieves a list of devices,
<http://login.etherios.com/ws/DeviceCore>

whereas this next URL retrieves a single device with ID 1.
<http://login.etherios.com/ws/DeviceCore/1>

4.2 Resource Web Service CRUD Conventions

The following CRUD conventions are used:

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

*A resource that has an ID that is generated by the database must be created using POST. Resources that have IDs that are composite values in the ID that are known can be created using a PUT.

4.2.1 Resource Overview

The following table lists all of the resources available using the Resource Web Service. The Get/Post/Put/Delete columns specify which operations are valid for the resource. The category column describes the typical usage of the web service:

DM - Device management

ED - Embedded device development

Resource Path	GET	POST	PUT	DELETE	Category	Description
DeviceCore	X	X	X	X	DM	Device and selected properties
DeviceInterface	X				DM	Device and network interface properties
DeviceMetaData	X	X	X	X	ED	Device descriptor data
DeviceVendor	X	X			ED	Embedded device developers
DeviceVendorSummary	X				ED	Device type list
FileData	X		X		DM	Data files
Group	X				DM	Organize devices into groups
NetworkInterface	X	X	X	X	DM	Device modem list
XbeeCore	X		X		DM	XBee nodes and properties

POST

HTTP POST is used to add resources to your account.

POST URL format is: /ResourcePath

Request content: XML 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 and 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. You may need a subscription.
 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](#) document with a root result element containing a location element for each resource created and any errors encountered.

GET

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

The GET [URI](#) format is:

/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 to use 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: It is recommended that you use the 'Accept-Encoding: gzip, deflate' request header. This instructs the server to return the data compressed with a return header of 'Transfer-Encoding: gzip'. This offers much better 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. You may need a subscription.
 500 (Internal server error) The request cannot be handled due to a server error. Wait and try again.

Return content:

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 returned -->
  <requestedSize>2</requestedSize>
  <!-- the remaining number of resources -->
  <remainingSize>0</remainingSize>
  <!-- ... List of the resources -->

```

Examples:

http://<hostname>/ws/DeviceCore will return all devices in the account matching the authorization credentials
 http://<hostname>/ws/DeviceCore/32 will return the device information matching the device where ID=32 (ID is an auto-generated number in Device Cloud)
 http://<hostname>/ws/DeviceCore?start=201&size=200 will return 200 records starting with record 201
 http://<hostname>/ws/DeviceCore?condition=devRecordStartDate>'2010-01-17T00:00:00Z' will return all devices added after midnight Jan 17th, 2010
 http://<hostname>/ws/DeviceCore/?condition=devConnectwareId='00000000-00000000-00409DFF-FF123456' will return 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>
    <csId>3</csId>
    <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>
  <csId>3</csId>
  <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>
  <dpContact />
  <dpDescription>Test device</dpDescription>
  <dpLocation />
  <dpServerId>ClientID[3]</dpServerId>
</DeviceCore>
</result>

```

PUT

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.

PUT URI format is: `/ResourcePath/ID`

Example: `/NetworkInterface/1`

Request content: XML 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) 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.

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

Return content: XML document with a root result element containing any errors encountered.

DELETE

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

DELETE URI format is: `/ResourcePath/ID`

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

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. You may need a subscription.

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

Return content: XML document with a root result element containing any errors encountered.

4.3 Compound Queries

Most of the examples of web services requests in this guide err on the simple side, in order to best show how each of the Web Services APIs work. However, in some cases, you may find that you need to set up compound queries. Here are a couple of demonstrations in how these will operate and what results you should expect:

- `https://<hostname>/ws/DiaChannelDataFull/?condition=devConnectwareID='00000000-00000000-00409DFF-FF123456' and ddInstanceName='transform0' and dcdUpdateTime>'2011-10-04T00:00:00Z'`
will return records for only device "00000000-00000000-00409DFF-FF123456", with ddInstanceName "transform0" updated after "2011-10-04T00:00:00Z"
- `https://<hostname>/ws/DiaChannelDataFull/.xls?condition=devConnectwareID='00000000-00000000-00409DFF-FF123456' and ddInstanceName='transform0' and dcdUpdateTime>'2011-10-04T00:00:00Z'`
will return records for only device "00000000-00000000-00409DFF-FF123456", with ddInstanceName "transform0" updated after "2011-10-04T00:00:00Z" and will deliver the results in Excel spreadsheet format (.xls).

NOTE: Modern browsers will MIME encode spaces (%20) and punctuation such as single quote (%27) on your behalf, but many programming languages will not. In those cases you will need to put them in proper MIME format in your code. For example, using the Python example shown earlier in this document, here is how a complex URL is encoded:

`webservice.putrequest("GET", "/ws/DiaChannelDataFull/?condition=devConnectwareID='%2700000000-00000000-00409DFF-FF123456%27%20' and %20ddInstanceName='%27transform0%27%20' and %20dcdUpdateTime%3E%272011-10-04T00:00:00Z%27")`

5. Resources

5.1 DeviceCore

The DeviceCore interface contains information specific to each device registered with Device Cloud. Some of these fields represent connection information while some select fields are taken from the device's settings or state information in order to make them easily searchable. In addition to using the "add device" capability in Device Cloud, this interface may be used to register devices. It may also be used to modify or delete devices from Device Cloud.

If you're specifying a group path which doesn't exist, the group path will be created for you.

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

HTTP operations supported:

- GET: Display a list of devices provisioned in your account
- POST: Add a device to your account
- PUT: Specify descriptive text fields for the device
- DELETE: Delete a device from your account

Elements include:

- id
 - devId - automatically generated ID for the device
 - devVersion - will be 0 which indicates it is the most recent version
- devRecordStartDate - date this record was created
- devMac - mac address of the device
- devCellularModemId - modem ID of the device
- devConnectwareId - Device ID of the device
- csId - the automatically generated ID assigned to a customer
- grpId - the automatically generated ID assigned to a customer's group
- grpPath - the name of the specified group
- devEffectiveStartDate - the date the device was provisioned in Device Cloud
- devTerminated - false if device is currently in the customer's account
- dvVendorId - ID of the vendor that manufactured the device
- dpDeviceType - manufacturer's device type such as ConnectPort X2
- dpFirmwareLevel - firmware as an integer such as 34209795
- dpFirmwareLevelDesc - firmware level as a string such as 2.10.0.3
- dpRestrictedStatus - restricts the device from connecting to Device Cloud (0: unrestricted, 2: restricted, 3: untrusted). This can be set using POST or PUT to ws/DeviceCore.
- dpLastKnownIp - IP address reported by the device such as 10.8.16.79
- dpGlobalIp - IP address from which the device connected
- dpConnectionStatus - 1 for connected, 0 for disconnected
- dpLastConnectTime - the date that the device last connected to Device Cloud such as 2010-07-21T15:20:00Z
- dpContact - contact setting from the device
- dpDescription - description setting from the device
- dpLocation - location setting from the device
- dpUserMetaData - this is a user definable free format text field
- dpTags - this is a comma delimited set of user defined tags
- dpPanId - panId setting from the device
- xpExtAddr - ZigBee extended address from the device
- dpMapLat - map latitude setting from the device
- dpMapLong - map longitude setting from the device
- dpServerId - an ID of the current server the device is connected to
- provisionId - pertains to the provisioning of a randomly generated ID. Must be used in place of devConnectwareId and your Vendor ID must be supplied.
- dpCurrentConnectPw - sets the password of the device. The device must send up this password when it connects to Device Cloud. The device will not be allowed to connect to Device Cloud without this password. See the [Security](#) Chapter for more information.

Examples:

Example #1:

Example for using DeviceCore with a PUT. The labels to be set (1) dpUserMetaData - this is a user definable free format text field; (2) dpTags - this is a comma delimited set of user defined tags.

```
<DeviceCore>
<!-- Either specify devConnectwareId or id -->
<devConnectwareId>00000000-00000000-00000000-00000000</devConnectwareId>
<!--
<id>
  <devId>0</devId>
</id>
-->
<!-- Specify one or both -->
<dpUserMetaData>This demo unit is located at 33 Main, Byron,MN</dpUserMetaData>
<dpTags>,demo,installed,</dpTags>
</DeviceCore>
```

Example #2:

To provision a device to a specific group issue the following request. The group will be created if it does not exist.

POST/ws/DeviceCore:

```
<DeviceCore>
<devConnectwareId>00000000-00000000-00000000-00000000</devConnectwareId>
<grpPath>test</grpPath>
</DeviceCore>
```

Example #3:

Retrieving a list of devices by group name can be done in one of two ways:

1. Issuing the following GET (substituting 'test' with your desired group name):
/ws/DeviceCore?condition=grpPath='test'
2. Issuing the following GET and text:
/ws/sci

```
<sci_request version="1.0">
  <send_message>
    <targets>
      <group path="test" />
    </targets>
    <rci_request version="1.1">
      <query_state />
    </rci_request>
  </send_message>
</sci_request>
```

Example #4:

In this example you will use DeviceCore to automatically generate a Device ID and add it to your account. This Device ID can then be used in a device that does not have a predefined Device ID. See Simplified HTTP Devices below.

This POST will create a new Device ID in your user account and sets the device password to "DevicePasswordHere."

In order for this to work, a vendor ID (121212 in this example) must be created for your user account using /ws/DeviceVendor. The grpPath must also be set to a valid group and you must verify that dpRestrictedStatus is set to the desired value in /ws/DeviceVendor.

POST/ws/DeviceCore:

```
<DeviceCore>
  <provisionId>121212</provisionId>
  <dpCurrentConnectPw>DevicePasswordHere</dpCurrentConnectPw>
</DeviceCore>
```

5.2 DeviceInterface

The DeviceInterface Web Service is used to retrieve information about your device(s) and associated network interfaces (i.e. cellular and satellite modems). This is a view combining the NetworkInterface table and a few select items from DeviceCore so that you may access the combined information in a single web service request.

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

HTTP operations supported:

- GET: Get a list of devices and associated network interfaces

Example:

Sample DeviceInterface result for 3 devices, the first has no network interfaces, the second has 1, the third has 2 for a total of 4 DeviceInterface records.

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <resultTotalRows>4</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>4</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <DeviceInterface>
    <id>
      <devId>144</devId>
      <devVersion>0</devVersion>
      <niId>0</niId>
      <niVersion>0</niVersion>
    </id>
    <devRecordStartDate>2010-01-21T19:19:00Z</devRecordStartDate>
    <devSerialNo>ABC123</devSerialNo>
    <devMac>00:40:9D:3C:1E:0F</devMac>
    <devConnectwareId>00000000-00000000-00409DFF-FF3C1E0F</devConnectwareId>
    <csId>7</csId>
    <grpId>7</grpId>
    <devEffectiveStartDate>2010-01-21T19:19:00Z</devEffectiveStartDate>
    <devTerminated>>false</devTerminated>
  </DeviceInterface>
  <DeviceInterface>
    <id>
      <devId>145</devId>
      <devVersion>0</devVersion>
      <niId>3</niId>
      <niVersion>0</niVersion>
    </id>
    <devRecordStartDate>2010-01-21T19:20:00Z</devRecordStartDate>
    <devSerialNo>ABC124</devSerialNo>
    <devMac>00:40:9D:3C:1E:4F</devMac>
    <devConnectwareId>00000000-00000000-00409DFF-FF3C1E4F</devConnectwareId>
    <csId>7</csId>
    <grpId>7</grpId>
    <devEffectiveStartDate>2010-01-21T19:20:00Z</devEffectiveStartDate>
    <devTerminated>>false</devTerminated>
    <niRecordStartDate>2010-01-21T19:52:00Z</niRecordStartDate>
    <niInterfaceType>1</niInterfaceType>
    <niSimId>8988216710502001122</niSimId>
    <niModemId>356021010924522</niModemId>
    <niEffectiveStartDate>2010-01-21T19:52:00Z</niEffectiveStartDate>
    <niTerminated>>false</niTerminated>
  </DeviceInterface>
  <DeviceInterface>
    <id>
      <devId>146</devId>
      <devVersion>0</devVersion>
      <niId>1</niId>
      <niVersion>0</niVersion>
    </id>
    <devRecordStartDate>2010-01-21T19:20:00Z</devRecordStartDate>
    <devSerialNo>ABC125</devSerialNo>
    <devMac>00:40:9D:3C:1E:5F</devMac>
    <devConnectwareId>00000000-00000000-00409DFF-FF3C1E5F</devConnectwareId>
    <csId>7</csId>
    <grpId>7</grpId>
    <devEffectiveStartDate>2010-01-21T19:20:00Z</devEffectiveStartDate>
    <devTerminated>>false</devTerminated>
    <niRecordStartDate>2010-01-21T19:43:00Z</niRecordStartDate>
    <niInterfaceType>2</niInterfaceType>
    <niModemId>SAT-modem-num</niModemId>
    <niEffectiveStartDate>2010-01-21T19:43:00Z</niEffectiveStartDate>
    <niTerminated>>false</niTerminated>
  </DeviceInterface>
  <DeviceInterface>
    <id>
      <devId>146</devId>
      <devVersion>0</devVersion>
      <niId>2</niId>
      <niVersion>0</niVersion>
    </id>
    <devRecordStartDate>2010-01-21T19:20:00Z</devRecordStartDate>
    <devSerialNo>ABC125</devSerialNo>
    <devMac>00:40:9D:3C:1E:5F</devMac>
    <devConnectwareId>00000000-00000000-00409DFF-FF3C1E5F</devConnectwareId>
    <csId>7</csId>
    <grpId>7</grpId>
    <devEffectiveStartDate>2010-01-21T19:20:00Z</devEffectiveStartDate>
    <devTerminated>>false</devTerminated>
    <niRecordStartDate>2010-01-21T19:51:00Z</niRecordStartDate>
    <niInterfaceType>1</niInterfaceType>
    <niSimId>8988216710502001110</niSimId>
    <niModemId>356021010924567</niModemId>
    <niEffectiveStartDate>2010-01-21T19:51:00Z</niEffectiveStartDate>
    <niTerminated>>false</niTerminated>
  </DeviceInterface>
</result>
```

5.3 DeviceMetaData

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

The DeviceMetaData cache contains many things available through [SCI](#). This web service call is used for the management of embedded device view descriptors which are not available directly from a device.

HTTP operations supported:

- GET: Display a list of view descriptors for any Vendor IDs that you own or are public
- POST: Add a view descriptor
- PUT: Update a view descriptor
- DELETE: Delete a view descriptor

Elements include:

- dmId - unique ID for this meta data
- dvVendorId - Vendor ID this meta data applies to
- dmDeviceType - device type this meta data corresponds to
- dmProductId - product ID this meta data corresponds to
- dmFirmwareId - firmware ID this meta data corresponds to
- dmVersion - firmware version this meta data corresponds to
- dmName - defines the type of descriptor, must be descriptor/ui
- dmCompressed - whether the meta data is stored compressed, typically false
- dmData - actual contents of the meta data

5.4 DeviceVendor

In order to get a Vendor ID, you must request one as described in the *Device Cloud User's Guide*.

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

HTTP operations supported:

- GET: Retrieve Vendor IDs that are available to you
- POST: Register for a Vendor ID to use for device development
- PUT: Update grpPath or dpRestrictedStatus

Elements include:

- dvVendorId - integer representation of this Vendor ID
- dvVendorIdDesc - hex representation of this Vendor ID
- cstId - Device Cloud ID of customer owning this Vendor ID
- dvDescription - information describing this Vendor ID
- dvRegistrationDate - date when the Vendor ID was registered
- grpPath - the name of a group into which new auto-provisioned devices will be provisioned by default. `<grpPath disabled="true"/>` will disable auto-provisioning. If you create a new Device ID by performing a POST to `ws/DeviceVendor` you can specify a `grpPath` that will override this default.
- dpRestrictedStatus - restricts the device from connecting to Device Cloud (0: unrestricted, 2: restricted, 3: untrusted). This can be set using POST or PUT to `ws/DeviceCore`.

5.4.1 Restriction levels

The following restriction levels are available for the `dpRestrictedStatus` element:

5.4.1.1 Unrestricted

- Everything is allowed

5.4.1.2 Restricted

- Disconnect and reboot are allowed
- Commands may be satisfied from the cache such as [SCI](#) requests specifying `cache="true"` or `cache="only"`.

5.4.1.3 Untrusted

Untrusted is used in cases where new devices are allowed to be auto-provisioned but the device should not be allowed to start working fully until it is validated as trusted. One example of how to validate a device is by having the device push up a application level user name and password, which the application would validate. Once accepted, the application will change the restriction status to "unrestricted".

- All device messaging is disabled except a device may push files with the specific file name 'untrusted'
- Users may make data service device requests to the device
- Disconnect and reboot are allowed
- Commands may be satisfied from the cache such as [SCI](#) requests specifying `cache="true"` or `cache="only"`.

Some operations that will be rejected include:

- All incoming SM messages for the device (including opt-in messages)
- All outgoing SM messages for the device (including request-connect)
- Redirect, Firmware Update, [RCI](#) Requests, File System Operations

5.5 DeviceVendorSummary

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

HTTP operations supported:

- GET: Provides a quick view of device types existing under your vendor ID

Elements include:

- dvVendorId - integer representation of the Vendor ID
- dmDeviceType - a device type existing under this Vendor ID
- dvVendorIdDesc - hex representation of the Vendor ID
- cstId - Device Cloud ID of customer owning this Vendor ID
- dvDescription - information describing this Vendor ID
- dmUiDescriptorCount - number of view descriptors that exist for this device type

5.6 FileData

Device Cloud provides an alternate web service to identify and retrieve data files from the server. Using this interface, clients can perform a general query to locate one or more files based upon metadata of the file such as its name, type, storage path, size, or modification date.

FileData is used by clients to read files reported by a device. Files will generally be available for 7 days; after that time, data will be pruned from the database. In order to retrieve file information past the 7 day limit you must set the archive flag during the initial upload. See Example #7 for more information on setting the archive flag.

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

HTTP operations supported:

- GET: Display a file or folder in your account
- PUT: Upload or change a file or folder in your account
- DELETE: Delete a file or folder from your account

Elements include:

- fdPath - indicates the hierarchical path to the file
- fdName - indicates the name of the file
- tld - indicates the unique Device Cloud ID of the customer owning the file
- CreatedDate - indicates the date that the file was first uploaded to Device Cloud
- LastModifiedDate - indicates the date that the file was last modified
- ContentType - indicates the type of data stored in the file
- Size - indicates the size of the file contents - in bytes
- Type - indicates the type of file (file or directory)
- [fdData] - indicates the Base64 encoded content of the file

Examples:

Example #1:

Fetching from FileData with no ID component or parameters will return a paged list of file metadata for all of your files.

GET `/ws/FileData`

Returns:

```
<?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>
    <csId>3439</csId>
    <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>
    <csId>3439</csId>
    <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 #2:

By specifying condition parameters, the caller can limit the files that are returned.

GET `/ws/FileData?condition=fdType='file' and fdLastModifiedDate>'2010-11-24T22:25:04Z'`

The above call returns all files written to Device Cloud after the specified date.

GET `/ws/FileData?condition=fdName like 'sample%25gas' and fdType='file' and fdLastModifiedDate>'2010-11-24T22:25:04Z'`

The above call returns all files whose name starts with 'sample' and ends with 'gas' that were written to Device Cloud after the specified date. For example, the sample above would match the filename 'sample3612gas.txt' but not filename 'sample3122water.txt'.

NOTE: '%25' is the URL-encoded representation of the percent sign (%).

Example #3:

To directly fetch the contents of a specific file, simply pass the path and name of the file as part of the URL like this:

GET `/ws/FileData/~00000000-00000000-00409DFF-FF640005/attribute_report.xml`

Returns:

```
<received_attribute_report timestamp="1297463631.0">
  <source_endpoint_id type="int">0x10</source_endpoint_id>
  <profile_id type="int">0x109</profile_id>
  <server_or_client type="int">0x0</server_or_client>
  <cluster_id type="int">0x702</cluster_id>
  <source_address type="MAC">00:40:9D:64:00:05:00:04</source_address>
  <record type="AttributeReportRecord">
    <attribute_id type="int">0x400</attribute_id>
    <attribute_type type="int">0x2A</attribute_type>
    <value type="int">0x13b</value>
  </record>
</received_attribute_report>
```

Example #4:

By specifying the option embed="true" the content of the files will be embedded in the results in Base64 format. For example, to get all files reported by a specific device that are newer than the specified

date:

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

Returns:

```
<?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>
```

Example #5:
To retrieve a specific file and view its contents:

GET ws/FileData/~/'yourFilePath/yourFileName.txt

Which returns the actual file contents such as:

This is the text of your test file!

Example #6:
To retrieve a file using conditions and to view its contents:

GET ws/FileData?condition=fdName='yourFileName.txt'&embed=true

Which returns the XML metadata with your file contents Base64 encoded in the fdData tags:

NOTE: If the filename is not unique you will get entries back for each matching file. This is beneficial if you are trying to fetch all of the files in a single read. In order to be more explicit you should also include the path.

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <resultTotalRows>1</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>1</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <FileData>
    <id>
      <fdPath>/yourFilePath/</fdPath>
      <fdName>yourFileName.txt</fdName>
    </id>
    <cstId>1</cstId>
    <fdCreatedDate>2010-10-01T14:24:20Z</fdCreatedDate>
    <fdLastModifiedDate>2010-10-08T17:09:48Z</fdLastModifiedDate>
    <fdContentType>text/plain</fdContentType>
    <fdSize>35</fdSize>
    <fdType>file</fdType>
    <fdData>VGhpcyBpcyB0aGUgdGV4dCBvZiB5b3VyIHRLc3QgZm1sZSE=</fdData>
  </FileData>
</result>
```

Example #7:
Files can be archived so that their history can be later retrieved through the /ws/FileDataHistory web service. To enable a file to be archived for future reference, set the 'archive' flag during the file upload. The content of the PUT is the actual contents of the file. If the archive flag is set, then in addition to the normal update to FileData, the POST or PUT operation will also store the new file content to FileDataHistory.

PUT http://<hostname>/ws/FileData/~/'test_folder/test.xml?type=file&archive=true

PUT on /ws/FileData/~/'test.xml

```
<FileData>
  <fdContentType>text/xml</fdContentType>
  <fdType>file</fdType>
  <fdData>SGVsbG8=</fdData>
  <fdArchive>true</fdArchive>
</FileData>
```

The 'fdData' field is Base64 encoded to represent Binary data in XML. A useful web tool to encode and decode Base64 data is available here: <http://ostermiller.org/calc/encode.html>.

SGVsbG8 is equivalent to hello

GET on /ws/FileData/~/'test.xml returns:

Hello

GET on /ws/FileDataHistory/~/'test.xml returns:

Hello

If you do the PUT again, GET on /ws/FileDataHistory/~/test.xml now returns:

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <resultTotalRows>2</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>2</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <FileDataHistory>
    <id>
      <fdPath>/db/CUS0000007_Digi_International/</fdPath>
      <fdName>test.xml</fdName>
      <fdhId>1902</fdhId>
    </id>
    <cstId>8</cstId>
    <fdCreatedDate>2011-09-20T22:00:50Z</fdCreatedDate>
    <fdLastModifiedDate>2011-09-20T22:00:50Z</fdLastModifiedDate>
    <fdContentType>text/xml</fdContentType>
    <fdSize>30</fdSize>
    <fdType>file</fdType>
  </FileDataHistory>
  <FileDataHistory>
    <id>
      <fdPath>/db/CUS0000007_Digi_International/</fdPath>
      <fdName>test.xml</fdName>
      <fdhId>1903</fdhId>
    </id>
    <cstId>8</cstId>
    <fdCreatedDate>2011-09-20T22:00:50Z</fdCreatedDate>
    <fdLastModifiedDate>2011-09-20T22:02:16Z</fdLastModifiedDate>
    <fdContentType>text/xml</fdContentType>
    <fdSize>5</fdSize>
    <fdType>file</fdType>
  </FileDataHistory>
</result>
```

To get the file content of each element, a parameter (embed) needs to be specified in the URL to say you want the raw data. This provides a Base64 encoded fdData field:

GET /ws/FileDataHistory/~test.xml?embed=true

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <resultTotalRows>2</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>2</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <FileDataHistory>
    <id>
      <fdPath>/db/CUS0000007_Digi_International/</fdPath>
      <fdName>test.xml</fdName>
      <fdhId>1902</fdhId>
    </id>
    <cstId>8</cstId>
    <fdCreatedDate>2011-09-20T22:00:50Z</fdCreatedDate>
    <fdLastModifiedDate>2011-09-20T22:00:50Z</fdLastModifiedDate>
    <fdContentType>text/xml</fdContentType>
    <fdSize>30</fdSize>
    <fdType>file</fdType>
    <fdData>QWxsIFlvdXIgQmFzZSB8cmUgQmVsb25nIFRvIFVz</fdData>
  </FileDataHistory>
  <FileDataHistory>
    <id>
      <fdPath>/db/CUS0000007_Digi_International/</fdPath>
      <fdName>test.xml</fdName>
      <fdhId>1903</fdhId>
    </id>
    <cstId>8</cstId>
    <fdCreatedDate>2011-09-20T22:00:50Z</fdCreatedDate>
    <fdLastModifiedDate>2011-09-20T22:02:16Z</fdLastModifiedDate>
    <fdContentType>text/xml</fdContentType>
    <fdSize>5</fdSize>
    <fdType>file</fdType>
    <fdData>SGVsbG8=</fdData>
  </FileDataHistory>
</result>
```

5.7 FileDataCore

Similar to a GET only version of FileData, use FileDataCore to get information about files stored on Device Cloud. Using this interface, you can expect to receive back a count and listing of all files and folders on your Device Cloud account. The main difference between a FileDataCore GET and a FileData GET is that a FileDataCore does not return file contents.

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

HTTP operations supported:

- GET: Display count, listing and basic information about files stored on Device Cloud

Example:

This result tells us that there are a total of eight files and/or folders. The first one is the root directory, in this case, a folder called CUS111000_Tech_Pubs, which is the unseen folder on the server for this Device Cloud account. The second result is a folder called 00000000-00000000-00409DFF-FF3C52EC, which is the folder automatically set up by Device Cloud for the device with that name. The very last result shows a file name of license.txt, which is stored in one of the device folders.

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <resultTotalRows>8</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>8</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <FileDataCore>
    <id>
      <fdPath>/db/</fdPath>
      <fdName>CUS111000_Tech_Pubs</fdName>
    </id>
    <cstId>27</cstId>
```

```

<fdArchive>false</fdArchive>
<fdContentType>application/xml</fdContentType>
<fdCreatedDate>2011-07-20T14:57:31Z</fdCreatedDate>
<fdLastModifiedDate>2011-07-20T14:57:31Z</fdLastModifiedDate>
<fdSize>0</fdSize>
<fdType>directory</fdType>
</FileDataCore>
<FileDataCore>
  <id>
    <fdPath>/db/CUS111000_Tech_Pubs</fdPath>
    <fdName>00000000-00000000-00409DFF-FF3C52EC</fdName>
  </id>
  <cstId>27</cstId>
  <fdArchive>false</fdArchive>
  <fdContentType>application/xml</fdContentType>
  <fdCreatedDate>2011-10-19T16:16:44Z</fdCreatedDate>
  <fdLastModifiedDate>2011-10-19T16:16:44Z</fdLastModifiedDate>
  <fdSize>0</fdSize>
  <fdType>directory</fdType>
</FileDataCore>
<FileDataCore>
  <id>
    <fdPath>/db/CUS111000_Tech_Pubs</fdPath>
    <fdName>00000000-00000000-00409DFF-FF3D6FB5</fdName>
  </id>
  <cstId>27</cstId>
  <fdArchive>false</fdArchive>
  <fdContentType>application/xml</fdContentType>
  <fdCreatedDate>2011-10-19T16:16:46Z</fdCreatedDate>
  <fdLastModifiedDate>2011-10-19T16:16:46Z</fdLastModifiedDate>
  <fdSize>0</fdSize>
  <fdType>directory</fdType>
</FileDataCore>
<FileDataCore>
  <id>
    <fdPath>/db/CUS111000_Tech_Pubs</fdPath>
    <fdName>00000000-00000000-00409DFF-FF4585E5</fdName>
  </id>
  <cstId>27</cstId>
  <fdArchive>false</fdArchive>
  <fdContentType>application/xml</fdContentType>
  <fdCreatedDate>2011-09-20T19:19:33Z</fdCreatedDate>
  <fdLastModifiedDate>2011-09-20T19:19:33Z</fdLastModifiedDate>
  <fdSize>0</fdSize>
  <fdType>directory</fdType>
</FileDataCore>
...most omitted for this example
<FileDataCore>
  <id>
    <fdPath>/db/CUS111000_Tech_Pubs/00000000-00000000-00409DFF-FF4A3946</fdPath>
    <fdName>license.txt</fdName>
  </id>
  <cstId>27</cstId>
  <fdArchive>false</fdArchive>
  <fdContentType>text/plain</fdContentType>
  <fdCreatedDate>2011-10-19T20:41:45Z</fdCreatedDate>
  <fdLastModifiedDate>2011-10-19T20:41:45Z</fdLastModifiedDate>
  <fdSize>0</fdSize>
  <fdType>file</fdType>
</FileDataCore>
</result>

```

5.8 FileDataHistory

FileDataHistory will display information from the archive for a file previously uploaded. In order to use this, the archive flag must already be set to force the file to be archived. Do this by setting a query parameter on the URL when uploading via FileData. See the [FileData](#) section for details.

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

HTTP operations supported:

- GET: Display history of activity for each file the user has uploaded to a device. This archive will only be available if the file has a flag set to archive to the history table when it's originally uploaded.

To see an example of how FileDataHistory works, it's best to see it used along with the FileData call. Knowing how to set up the files appropriately is an important piece of understanding FileDataHistory. See Example #7 within the [FileData](#) section for more information.

5.9 Group

The Group Web Service is used to retrieve information about groups.

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

HTTP operations supported:

- GET: Get a list of groups

Elements include:

- grpId - integer representation of the Group
- grpName - name of the Group
- grpDescription - description of the Group
- grpPath - full path of the Group
- grpParentId - integer representation of the Group's parent

5.10 NetworkInterface

NetworkInterface contains specific information related to external network interfaces for devices where Device Cloud needs to have knowledge of that information in order to interact with those devices. For example, Device Cloud 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>

HTTP operations supported:

- GET: Display a list of modems provisioned in your account
- POST: Add a modem to your account
- PUT: Update modem information in your account
- DELETE: Delete a modem from your account

Elements include:

- id
 - nild - automatically generated ID for the NetworkInterface
 - niVersion - will be 0 which indicates it is the most recent version of the NetworkInterface record
- cstld - the automatically generated ID assigned to a customer
- grpId - the automatically generated ID assigned to a customer's group
- devId - the ID of the device this NetworkInterface record is associated with
- devVersion - will be 0, which indicates the most recent version of the device record
- niInterfaceType - the type of network interface (0: None, 1: GSM, 2: CDMA, 3: ORBComm, 4: Iridium)
- niSimId - the ICCID, MEID, or ESN of the SIM or cellular modem
- niModemId - the modem ID of the satellite modem
- niPhone - the phone number of the cellular line. Should be in international form (e.g. +1 123-456-7890).
- niActivePhone - true if this record contains the phone number (niPhone) to which Device Cloud will send SMS messages to for this device. Only one NetworkInterface record can have niActivePhone set to true per device.
- nildigiPhone - the short code or long code that the device uses to communicate with Device Cloud
- nildigiServiceId - the keyword used in the shared code
- nilmsi - the IMSI (International Mobile Subscriber Identity) of the SIM

NetworkInterface records typically fall into two possible categories: cellular lines or satellite modems. When creating a new NetworkInterface record via a POST, different fields are required depending on the type of NetworkInterface what Device Cloud features are going to be used with it. All NetworkInterface records require the devId and niInterfaceType fields to be set. Additionally, the following fields are required depending on what features are going to be used:

For Carrier Integration:

- niInterfaceType needs to be set to 1 (GSM) or 2 (CDMA).
- niSimId needs to be set if the cellular line is with AT&T, Deutsche Telekom, Rogers, Telefonica or Verizon.
- nilmsi needs to be set if the cellular line is with Vodafone.

For SMS Messaging:

- niPhone needs to be set so that Device Cloud knows which phone number to send SMSs to for the device.

For Satellite Messaging:

- niInterfaceType needs to be set to 3 (ORBComm) or 4 (Iridium). Currently, only Iridium is supported with Device Cloud satellite messaging.
- niModemId needs to be set to the modem ID of the device's satellite modem.

You can create or update NetworkInterface records with this information via a web services request. For SMS-capable devices see the [Configure Device Cloud with the Phone Number of the Device](#) section for more information. For Iridium satellite-capable devices see the [Configure Device Cloud with the Modem ID of the Device](#) section for more information.

5.11 XbeeCore

Similar to DeviceCore, this interface contains information about the mesh network associated with each device registered in Device Cloud. These records are organized so they may be easily searchable by information specific to each node, regardless of which device they are associated with. These records are updated whenever an XBee discovery is performed on the device to which these nodes are associated. The caller can send an Xbee discovery to the gateway by specifying either the *cache=false* option or *clear=true* as a parameter to the XbeeCore GET request. The *cache=false* parameter requests the current list of the discovered XBee nodes from the gateway, while *clear=true* clears the current list of nodes and performs a discover to get fresh results. Note that the discover requests need to be directed to a single gateway.

URI: `http://<hostname>/ws/XbeeCore`

HTTP operations supported:

- GET: Display a list of ZigBee nodes in your account
- PUT: Specify descriptive text field for the node

Elements include:

- cstld - Device Cloud ID of customer owning the gateway discovering the node
- grpId - Device Cloud ID of the customer group owning the gateway
- grpPath - the name of the specified group
- devConnectwareId - Device ID of the gateway discovering this attributes node
- xpExtAddr - ZigBee 64bit extended address of node
- xpNetAddr - ZigBee 16bit network address of the node
- xpNodeType - ZigBee node (device) type (0=coordinator, 1=router, 2=endnode)
- xpParentAddr - If node is an endnode this will be the network addr of the router it connects to. If node is a router this will be 0xFFFF.
- xpProfileId - ZigBee device profile id of the node
- xpMfgId - ZigBee manufacturing id of the node
- xpDeviceType - Device type of the node. Low order 16 bits indicates the product type. High order 16 bits indicates the module type. For convenience text descriptions are returned in xmtModuleTypeDesc and xptProductTypeDesc fields.
- xpNodeId - ZigBee node identifier
- xpDiscoveryIndex - Index within the list of nodes discovered on the HAN
- xmtModuleTypeDesc - Text description of the module type defined in xpDeviceType
- xptProductTypeDesc - Text description of the product type defined in xpDeviceType
- xpStatus - 1 for connected, 0 for disconnected. (Note: xpStatus is available only for Smart Energy nodes)
- xpUpdateTime - time the node was last discovered
- xpUserMetaData - this is a user definable free format text field

Examples:

To query the nodes on a specific HAN (gateway device ID 00000000-00000000-00409DFF-FF702005):

GET `ws/XbeeCore?condition=devConnectwareId='00000000-00000000-00409DFF-FF702005'`

Returns:

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <resultTotalRows>3</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>3</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <XbeeCore>
    <xpExtAddr>00:13:A2:00:40:23:22:01</xpExtAddr>
    <devConnectwareId>00000000-00000000-00409DFF-FF702005</devConnectwareId>
    <cstId>3</cstId>
    <grpId>3</grpId>
```

```

<xpNetAddr>3</xpNetAddr>
<xpNodeType>1</xpNodeType>
<xpParentAddr>65534</xpParentAddr>
<xpProfileId>49413</xpProfileId>
<xpMfgId>4126</xpMfgId>
<xpDeviceType>196608</xpDeviceType>
<xpNodeId>thermostat</xpNodeId>
<xpDiscoveryIndex>3</xpDiscoveryIndex>
<xpStatus>1</xpStatus>
<xmtModuleTypeDesc>XBee ZB</xmtModuleTypeDesc>
<xptProductTypeDesc>Unspecified</xptProductTypeDesc>
<xpUpdateTime>2009-11-01T00:00:00Z</xpUpdateTime>
</XbeeCore>
<XbeeCore>
  <xpExtAddr>00:13:A2:00:40:27:03:93</xpExtAddr>
  <devConnectwareId>00000000-00000000-00409DFF-FF702005</devConnectwareId>
  <csId>3</csId>
  <grpId>3</grpId>
  <xpNetAddr>1</xpNetAddr>
  <xpNodeType>1</xpNodeType>
  <xpParentAddr>65534</xpParentAddr>
  <xpProfileId>49413</xpProfileId>
  <xpMfgId>4126</xpMfgId>
  <xpDeviceType>196611</xpDeviceType>
  <xpNodeId>aux gw</xpNodeId>
  <xpDiscoveryIndex>1</xpDiscoveryIndex>
  <xpStatus>1</xpStatus>
  <xmtModuleTypeDesc>XBee ZB</xmtModuleTypeDesc>
  <xptProductTypeDesc>X2 Gateway</xptProductTypeDesc>
  <xpUpdateTime>2009-11-01T00:00:00Z</xpUpdateTime>
</XbeeCore>
<XbeeCore>
  <xpExtAddr>00:13:A2:00:40:66:33:03</xpExtAddr>
  <devConnectwareId>00000000-00000000-00409DFF-FF702005</devConnectwareId>
  <csId>3</csId>
  <grpId>3</grpId>
  <xpNetAddr>0</xpNetAddr>
  <xpNodeType>0</xpNodeType>
  <xpParentAddr>65534</xpParentAddr>
  <xpProfileId>49413</xpProfileId>
  <xpMfgId>4126</xpMfgId>
  <xpDeviceType>196608</xpDeviceType>
  <xpNodeId>esp meter</xpNodeId>
  <xpDiscoveryIndex>2</xpDiscoveryIndex>
  <xpStatus>1</xpStatus>
  <xmtModuleTypeDesc>XBee ZB</xmtModuleTypeDesc>
  <xptProductTypeDesc>Unspecified</xptProductTypeDesc>
  <xpUpdateTime>2009-11-01T00:00:00Z</xpUpdateTime>
</XbeeCore>
</result>

```

To request the current list of nodes from a gateway (device ID 00000000-00000000-00409DFF-FF702005):

GET ws/XbeeCore?condition=devConnectwareId='00000000-00000000-00409DFF-FF702005'&cache=false

To request a fresh discovery of a gateway's nodes:

GET ws/XbeeCore?condition=xpExtAddr='00:13:A2:00:00:00:00:00'&clear=true

6. SCI (Server Command Interface)

6.1 Introduction

SCI (Server Command Interface) is a web service that allows users to access information and perform commands that relate to their device. Examples of these requests include:

1. Retrieve live or cached information about your device(s)
2. Change settings on your device(s)
3. Interact with a Python program running on your device(s) to send commands or retrieve information
4. Read from and write to the file system on your device(s)
 - Update your Python applications
 - Retrieve data stored locally on your device(s)
5. Update device firmware
6. Update XBee radio firmware on your device(s)
7. 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 will 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`.

6.2 Anatomy of an 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

6.2.1 File Reference

The payload for an SCI command can be referenced from a file in Device Cloud Data Services as opposed to being explicitly described in the actual request. For example, the following SCI request's payload is being 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>
```

6.3 Targets

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

1. <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.
2. <device id="all"/>
When included in an SCI request, this element specifies the device IDs of all of your Device Cloud-registered devices. Requests issued will be sent to all of the devices registered within your Device Cloud user account.
3. <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.
4. <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.

6.4 Synchronous Request

To send a synchronous request using a device ID:

POST the following to: <http://login.etherios.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 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 will return 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: <http://login.etherios.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 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://login.etherios.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.

6.5 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
```

6.5.1 Performing an Asynchronous Request

Asynchronous request is performed by specifying `synchronous="false"` in the elements 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.

6.5.2 Retrieve Status

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

6.5.2.1 Status for a Particular Job

Do an HTTP GET on `http://login.etherios.com/ws/sci/{job_id}`

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

6.5.2.2 Overall Status of Outstanding Jobs

Do an HTTP GET on `http://login.etherios.com/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>
    <csiId>0</csiId>
    <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.

6.5.3 Retrieve Progress

You can retrieve the progress for a particular SCI job by specifying the `progress=true` parameter, i.e.:

HTTP GET `http://login.etherios.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.

6.5.4 Cancel a Request or Delete the Results

Do an HTTP DELETE on `http://login.etherios.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.

6.6 Available Operations

Several operations are currently available.

- **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.
- **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 Device Cloud will execute the command right away. If the device is not connected, then Device Cloud will execute the command as soon as the device connects to Device Cloud. 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 Device Cloud. 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 Device Cloud, 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.

6.6.1 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. Device Cloud 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>
```

6.6.1.1 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.

6.6.1.2 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.

6.6.1.3 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>
```

6.6.1.4 reboot

Reboots device immediately.

6.6.1.5 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.

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.

Is

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 $\geq$ start. For some reason, if $\text{start} > 1$, 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>
```

6.6.2 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 Device Cloud, 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 Device Cloud completed successfully.

It is still possible for the process to fail between Device Cloud 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="00000000-00000000-00000000-00000000">
      <submitted />
    </device>
  </update_firmware>
</sci_reply>
```

To do the update operation with a file stored in Device Cloud 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.

6.6.3 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>

```

6.6.4 query_firmware_targets

QueryFirmware 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>

```

6.6.5 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:

6.6.5.1 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>
```

6.6.5.2 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>
```

6.6.5.3 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, and crc32. any is used to indicate the device should choose its best available hash. md5 or crc32 may be specified but the device may not support them (or possibly any hash mechanism at all).

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>
```

6.6.5.4 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>
```

6.6.6 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>
<result>
```

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>
```

6.6.7 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>
  <requests>
    <device_request target_name="myTarget">my payload string</device_request>
  </requests>
</reboot>
```

7. Data Files

7.1 Introduction

The Data Web Service provides a temporary repository for storing files for later retrieval by either the device or web services application. The most common usage is for devices to post data to the data store autonomously so that a web services client application may check periodically to retrieve any new or updated contents.

```
URI: http://<hostname>/ws/data
```

The server is listening for requests with a path of /ws/data and the server treats the rest of the path as the path to a database collection or file. The root of each customers collection is always /db/{customer identifier}. This home collection of an authenticated user's customer account is aliased with a tilde (~). As an example, to access the mydata collection under the user's home collection they could specify a URL like this:

```
http://<hostname>/ws/data/~mydata
```

GET	Retrieves a representation of a file or collection from the database.
HEAD	Gets information about a file or collection from the database.
PUT	Uploads a file to the database. If a collection does not exist, the collection will be created and the file will be added to the newly created collection.
DELETE	Removes a document or collection from the database.
POST	Submits data in the form of an XML fragment in the content of the request which specifies the action to take. POST can also be used to upload a file to the database (instead of PUT). This is useful in standard web browser applications which don't have the ability to do a PUT. Use the standard multipart/form-data encoding type in a form to upload a file with the post method. Refer to the POST section below for more information.

PUT
Put a file or a folder to storage.

PUT URI format is: http://<hostname>/ws/data/~/{data path}[?_type={file | folder}]

Examples:

```
/ws/data/~test?_type=folder
/ws/data/~test/test.xml?_type=file
/ws/data/~test/test.xml
```

Request content: Ignored for folder, file contents for a file

Return codes:
201 (Created)
400 (Bad request) if the request is invalid
403 (Forbidden) if the request is an invalid path for the user
503 (Service unavailable) if the storage service is not available

Response header: No added data

Return content: None

GET
Get a file or a folder listing from storage.

GET URI format is: /ws/data/{data path}[?_recursive={yes | no}]

Examples:

```
/ws/data/~test?_recursive=yes
/ws/data/~test/test.xml
```

Request content: none

Return codes:
200 (OK)
400 (Bad request) if the request is invalid
403 (Forbidden) if the request is an invalid path for the user
404 (Not found) if the path is not found
501 (Not implemented) if the request cannot be handled because the query parameter value is not implemented.
503 (Service unavailable) if the storage service is not available

Response header: sets Content-Type, Last-Modified, CacheControl, Expires. For file, also sets content length

Return content: List for a folder, contents for a file

HEAD

Get a file or a folder information from storage.

HEAD URI format is: /ws/data/{data path}[?_recursive={yes | no}]

Examples:

```
/ws/data/~ /test?_recursive=yes  
/ws/data/~ /test/test.xml
```

Request content: none

Return codes:

200 (OK)
400 (Bad request) if the request is invalid
403 (Forbidden) if the request is an invalid path for the user
404 (Not found) if the path is not found
501 (Not implemented) if the request cannot be handled because the query parameter value is not implemented
503 (Service unavailable) if the storage service is not available

Response header: sets Content-Type, Last-Modified, Content-Length

Return content: None

DELETE

Delete a file or a folder from storage.

DELETE URI format is: /ws/data/{data path}

Examples:

```
/ws/data/~ /test  
/ws/data/~ /test/test.xml
```

Request content: none

Return codes: 200 (OK)

400 (Bad request) if the request is invalid
403 (Forbidden) if the request is an invalid path for the user
404 (Not found) if the path is not found
503 (Service unavailable) if the storage service is not available

Response header: No added data

Return content: None

POST

Put a multipart file to storage.

POST URI format is: /ws/data /{data path}

Example:

```
/ws/data/~ /test/test.xml
```

Request content: Multi-part file contents for a file. The first part can be a _responseformat of json if json return content is requested.

Return codes:

201 (Created)
400 (Bad request) if the request is invalid including not a multipart file
403 (Forbidden) if the request is an invalid path for the user
503 (Service unavailable) if the storage service is not available

Response header: content-type

Return content: json or xml message data

8. Security

In order to verify the identity of a device, Device Cloud supports a device password to be used with EDP.

Setting an EDP password for a device will cause Device Cloud to verify the identity of the device on connect. If a device has a password set in Device Cloud, it must be configured to send up that password, or Device Cloud will not allow the connection. By default, Device Cloud is configured without a password for a device. This means any device with that ID will be allowed to connect, regardless of whether it supplies a password or not.

8.1 Initial Password

When provisioning a device you may specify an initial password. Note that the device must already have been configured with this password. This must be done through the DeviceCore Web Service, by including an additional field dpCurrentConnectPw. If the field is omitted, the provisioned device will default to no password. The content for provisioning a device with a password through DeviceCore would look like:

```
<DeviceCore>  
  <devMac>00:40:9D:22:22:21</devMac>  
  <dpCurrentConnectPw>1234</dpCurrentConnectPw>  
</DeviceCore>
```

See the [DeviceCore](#) section for more information on provisioning a device.

8.2 Changing a Password

Once a device has been provisioned, a password for the device can be set or removed.

By setting the password for a device, the server will attempt to configure the device to use the new password. Until the server is able to successfully configure the device, it will be allowed to connect with the previous password. Once the server has configured the device, it will require the device to connect with that password for subsequent connections.

Removing a password will remove it from the server, which will cause the password of the device to no longer be verified on connect. It does not cascade to the device, and as a result, the device will still be configured to use a password. The device will still be able to connect however, as the server will simply ignore the password if it is not configured to use one.

Setting or removing a password is done through the Security Web Service, as shown below.

POST URI format is: /ws/security

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>
```

Multiple device elements may be specified.

Return codes:

200 (OK) Request is complete

207 (Multi-status) A failure occurred when processing the request.

400 (Bad request) The request is invalid.

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

Return content: XML document with a root set_password or remove_password element specifying the original command, and a list of device elements each containing child errors if the request could not be processed for that device.

9. Energy

9.1 Manufacturer Specific Attributes

Many of the Energy API's contain an xaAttributeId field. This field holds the attribute ID. These are typically standard identifiers defined for a cluster. It is possible, however, for a device to define non-standard, manufacturer specific attributes. These may be defined on existing clusters or new clusters. To distinguish manufacturer specific attributes from standard attributes a manufacturer code is assigned to them. The attribute ID is placed in the low order 4 bytes of the xaAttributeId field, and the manufacturer specific code is placed in the high order 4 bytes. For example, the standard Smart Energy simple metering cluster (0x702) assigns an attribute ID of 0x2 for the Current Max Demand Delivered. A manufacturer specific attribute ID of 0x2 could also be assigned to the simple meter cluster. If the manufacturer code is 0x101E, then the xaAttributeId field for the manufacturer specific attribute would be 0x0000101E00000002 (or 17721035063298 decimal).

The following table lists the Energy APIs available using the Resource Web Services. The Get/Post/Put/Delete columns specify which operations are valid for the resource.

9.2 Energy APIs

Resource Path	GET	POST	PUT	DELETE	Description
XbeeAttributeCore	X				XBee Attribute Definitions
XbeeAttributeFull	X				XBee Attribute Definitions
XbeeAttributeDataCore	X		X	X	Current XBee Attribute Values
XbeeAttributeDataFull	X		X	X	Current XBee Attribute Values
XbeeAttributeDataHistoryCore	X			X	Timestamped XBee Attribute Values
XbeeAttributeDataHistoryFull	X			X	Timestamped XBee Attribute Values
XbeeAttributeReportingCore	X		X	X	SE Attribute Reporting Config. - deprecated 2.9.0
XbeeClusterCore	X				XBee Endpoint and Cluster Definitions
XbeeEventDataCore	X		X	X	Current SE Event Data
XbeeEventDataFull	X		X	X	Current SE Event Data
XbeeEventDataHistoryCore	X			X	Timestamped SE Event Data
XbeeEventDataHistoryFull	X			X	Timestamped SE Event Data

9.3 XbeeAttributeCore

This API is used by clients to identify one or more attributes of any node in their set of HAN networks. Unlike XbeeAttributeDataCore, which includes the data associated with the attributes, this interface is a good way to retrieve a list of attributes that are available on any or all nodes that you have access to.

GET ws/XbeeAttributeCore[?param1¶m2...¶mn]

This API is used to query portions of the available HAN network topology. Callers can scope the returned information through the use of condition parameters.

Elements include:

- devConnectwareId - Connectware ID of the gateway discovering this attributes node
- xpExtAddr - Zigbee 64bit extended address of node
- xpParentAddr - If node is an endnode this will be the ext addr of the router it connects to. If node is a router this will be 0xFFFF.
- xeEndpointId - Zigbee endpoint this attribute resides on
- xeProfileId - associated Zigbee profile
- xeDeviceId - associated Zigbee device type
- xeDeviceVersion - associated Zigbee device version
- xcClusterId - associated Zigbee cluster
- xcClusterType - kind of associated Zigbee cluster (0=server or 1=client)
- xaAttributeId - Zigbee attribute ID
- xaAttributeType - Zigbee type of the attribute (see zcl and associated profile spec for available types)

GET Examples:

To query all the cluster/attributes supported on a specific node of a specific HAN:

GET ws/XbeeAttributeCore?condition=xpExtAddr='00:13:A2:00:40:66:33:03'

which returns:

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <resultTotalRows>15</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>15</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <XbeeAttributeCore>
    <id>
      <xpExtAddr>00:13:A2:00:40:66:33:03</xpExtAddr>
      <xeEndpointId>1</xeEndpointId>
      <xcClusterType>0</xcClusterType>
      <xcClusterId>1794</xcClusterId>
      <xaAttributeId>0</xaAttributeId>
    </id>
  </XbeeAttributeCore>
</result>
```

```

<csId>3</csId>
<devConnectwareId>00000000-00000000-00000000-00000000</devConnectwareId>
<xpParentAddr>65534</xpParentAddr>
<xeProfileId>265</xeProfileId>
<xeDeviceId>1280</xeDeviceId>
<xeDeviceVersion>1</xeDeviceVersion>
<xaAttributeType>37</xaAttributeType>
</XbeeAttributeCore>
<XbeeAttributeCore>
  <id>
    <xpExtAddr>00:13:A2:00:40:66:33:03</xpExtAddr>
    <xeEndpointId>1</xeEndpointId>
    <xcClusterType>0</xcClusterType>
    <xcClusterId>1794</xcClusterId>
    <xaAttributeId>256</xaAttributeId>
  </id>
  <csId>3</csId>
  <devConnectwareId>00000000-00000000-00000000-00000000</devConnectwareId>
  <xpParentAddr>65534</xpParentAddr>
  <xeProfileId>265</xeProfileId>
  <xeDeviceId>1280</xeDeviceId>
  <xeDeviceVersion>1</xeDeviceVersion>
  <xaAttributeType>37</xaAttributeType>
</XbeeAttributeCore>
...
<XbeeAttributeCore>
  <id>
    <xpExtAddr>00:13:A2:00:40:66:33:03</xpExtAddr>
    <xeEndpointId>1</xeEndpointId>
    <xcClusterType>0</xcClusterType>
    <xcClusterId>1794</xcClusterId>
    <xaAttributeId>1280</xaAttributeId>
  </id>
  <csId>3</csId>
  <devConnectwareId>00000000-00000000-00000000-00000000</devConnectwareId>
  <xpParentAddr>65534</xpParentAddr>
  <xeProfileId>265</xeProfileId>
  <xeDeviceId>1280</xeDeviceId>
  <xeDeviceVersion>1</xeDeviceVersion>
  <xaAttributeType>32</xaAttributeType>
</XbeeAttributeCore>
</result>

```

9.4 XbeeAttributeFull

URI: <http://<hostname>/ws/ XbeeAttributeFull>

HTTP operations supported:

- GET: Display a list of ZigBee attribute names - the Full variation contains more data than the Core

9.5 XbeeAttributeDataCore

XbeeAttributeData is used by clients to read and update attribute values of a node/endpoint/cluster in a HAN network. Use the *start_reports* command of the Smart Energy Gateway to schedule automatic reporting of specific attribute values. This reporting can be based on time or change intervals. As these attributes are reported, they are timestamped and saved in the server for later retrieval through this interface. To query current attribute values use the XbeeAttributeDataCore Full controller. To query historical readings use the XbeeAttributeDataHistoryCore Full controller and provide a time constraint.

XbeeAttributeDataCore is used to query reported usage data about one or more devices and their attributes. Callers can query for current attribute values using the normal condition options. In addition, callers can query previously reported values by including xadUpdateTime constraints in the condition expression. Callers can specify an aggregate constraint to perform special operations against the matching result set. The aggregate operations are: SUM, AVG, MAX, MIN, COUNT. Finally, the caller can force a deep reading of the current value (by sending a ZCL command and waiting for the reply) by specifying the *cache=false* option.

URI: <http://<hostname>/ws/ XbeeAttributeDataCore>

HTTP operations supported:

- GET: Display a list of the most recent ZigBee attribute values
- PUT: Update writable attribute information about a node
- DELETE

Elements include:

- csId - Device Cloud ID of customer owning the gateway discovering this attributes node
- devConnectwareId - Device ID of the gateway discovering this attributes node
- xpExtAddr - ZigBee 64bit extended address of node
- xpParentAddr - If node is an endnode this will be the ext addr of the router it connects to. If node is a router this will be 0xFFFF.
- xeEndpointId - ZigBee endpoint this attribute resides on
- xeProfileId - associated ZigBee profile
- xeDeviceId - associated ZigBee device type
- xeDeviceVersion - associated ZigBee device version
- xcClusterId - associated ZigBee cluster
- xcClusterType - kind of associated ZigBee cluster (0=server or 1=client)
- xaAttributeId - ZigBee attribute ID
- xaAttributeType - ZigBee type of the attribute (see zcl and associated profile spec for available types)
- xadAttributeStringValue - string form of the value (always available)
- xadAttributeIntegerValue - type specific form of value (only available for corresponding ZigBee attribute types, else null)
- xadAttributeFloatValue - type specific form of value (only available for corresponding ZigBee attribute types, else null)
- xadAttributeBooleanValue - type specific form of value (only available for corresponding ZigBee attribute types, else null)
- xadAttributeDateValue - type specific form of value (only available for corresponding ZigBee attribute types, else null)
- xadUpdateTime - time that the attribute value was last reported or set

GET Examples:

To query the latest CurrentSummationDelivered reading for a specific simple meter:

GET <ws/XbeeAttributeDataCore?condition=xpExtAddr='00:13:A2:00:40:66:33:03' and xeEndpointId= 1 and xcClusterType= 0 and xcClusterId= 1794 and xaAttributeId= 0>

which returns:

```

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

```

```

<requestedSize>1000</requestedSize>
<remainingSize>0</remainingSize>
<XbeeAttributeDataCore>
  <id>
    <xpExtAddr>00:13:A2:00:40:66:33:03</xpExtAddr>
    <xeEndpointId>1</xeEndpointId>
    <xcClusterType>0</xcClusterType>
    <xcClusterId>1794</xcClusterId>
    <xaAttributeId>0</xaAttributeId>
  </id>
  <csId>3</csId>
  <devConnectwareId>00000000-00000000-00000000-00000000</devConnectwareId>
  <xpParentAddr>65534</xpParentAddr>
  <xeProfileId>265</xeProfileId>
  <xeDeviceId>1280</xeDeviceId>
  <xeDeviceVersion>1</xeDeviceVersion>
  <xaAttributeType>37</xaAttributeType>
  <xadAttributeStringValue>5017</xadAttributeStringValue>
  <xadAttributeIntegerValue>5017</xadAttributeIntegerValue>
  <xadUpdateTime>2010-06-14T16:21:43Z</xadUpdateTime>
</XbeeAttributeDataCore>
</result>

```

To query the total latest meter reading data for all simple meters in a specific HAN:

GET ws/XbeeAttributeDataCore?condition=devConnectwareId='00000000-00000000-00000000-00000000' and xeProfileId=265 and xcClusterType= 0 and xcClusterId= 1794 and xaAttributeId= 0&aggregate=SUM

returns:

```

<result>
  <resultTotalRows>1</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>1</resultSize>
  <requestedSize>1</requestedSize>
  <remainingSize>0</remainingSize>
  <XbeeAttributeDataCore>
    <sum>21156</sum>
  </XbeeAttributeDataCore>
</result>

```

To query the total number of simple meters currently online:

GET ws/XbeeAttributeDataCore?condition=xeProfileId=265 and xeDeviceId=1280&aggregate=COUNT

returns:

```

<result>
  <resultTotalRows>1</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>1</resultSize>
  <requestedSize>1</requestedSize>
  <remainingSize>0</remainingSize>
  <XbeeAttributeDataCore>
    <count>2</count>
  </XbeeAttributeDataCore>
</result>

```

PUT examples:

To set the OccupiedCoolingSetpoint of a specific thermostat in the HAN:

PUT ws/XbeeAttributeDataCore

body:

```

<XbeeAttributeDataCore>
  <id>
    <xpExtAddr>00:13:A2:00:40:23:22:01</xpExtAddr>
    <xeEndpointId>1</xeEndpointId>
    <xcClusterType>0</xcClusterType>
    <xcClusterId>513</xcClusterId>
    <xaAttributeId>17</xaAttributeId>
  </id>
  <devConnectwareId>00000000-00000000-00000000-00000000</devConnectwareId>
  <xeProfileId>265</xeProfileId>
  <xeDeviceId>1283</xeDeviceId>
  <xeDeviceVersion>1</xeDeviceVersion>
  <xaAttributeType>37</xaAttributeType>
  <xadAttributeStringValue>78</xadAttributeStringValue>
</XbeeAttributeDataCore>

```

9.6 XbeeAttributeDataFull

URI: http://<hostname>/ws/ XbeeAttributeDataFull

HTTP operations supported:

- GET: Display a list of the most recent ZigBee attribute values - the Full variation contains more data than the Core
- PUT
- DELETE

9.7 XbeeAttributeDataHistoryCore

URI: http://<hostname>/ws/ XbeeAttributeDataHistoryCore

HTTP operations supported:

- GET: Display a list of the historical ZigBee attribute values
- DELETE

Example:

To query all the CurrentSummationDelivered readings for a specific simple meter since a specific time query the History controller with the desired time constraint:

GET ws/XbeeAttributeDataHistoryCore?condition=xpExtAddr='00:13:A2:00:40:66:33:03' and xeEndpointId= 1 and xcClusterType= 0 and xcClusterId= 1794 and xaAttributeId= 0 and

xadUpdateTime>2010-01-16T18:00:00Z'

returns:

```
<result>
  <resultTotalRows>3</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>3</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <XbeeAttributeDataHistoryCore>
    <id>
      <xpExtAddr>00:13:A2:00:40:66:33:03</xpExtAddr>
      <xeEndpointId>1</xeEndpointId>
      <xcClusterType>0</xcClusterType>
      <xcClusterId>1794</xcClusterId>
      <xaAttributeId>0</xaAttributeId>
      <xadhId>615</xadhId>
    </id>
    <csId>3</csId>
    <devConnectwareId>00000000-00000000-00000000-00000000</devConnectwareId>
    <xpParentAddr>65534</xpParentAddr>
    <xeProfileId>265</xeProfileId>
    <xeDeviceId>1280</xeDeviceId>
    <xeDeviceVersion>1</xeDeviceVersion>
    <xaAttributeType>37</xaAttributeType>
    <xadAttributeStringValue>5008</xadAttributeStringValue>
    <xadAttributeIntegerValue>5008</xadAttributeIntegerValue>
    <xadUpdateTime>2010-06-14T15:51:42Z</xadUpdateTime>
  </XbeeAttributeDataHistoryCore>
  <XbeeAttributeDataHistoryCore>
    <id>
      <xpExtAddr>00:13:A2:00:40:66:33:03</xpExtAddr>
      <xeEndpointId>1</xeEndpointId>
      <xcClusterType>0</xcClusterType>
      <xcClusterId>1794</xcClusterId>
      <xaAttributeId>0</xaAttributeId>
      <xadhId>651</xadhId>
    </id>
    <csId>3</csId>
    <devConnectwareId>00000000-00000000-00000000-00000000</devConnectwareId>
    <xpParentAddr>65534</xpParentAddr>
    <xeProfileId>265</xeProfileId>
    <xeDeviceId>1280</xeDeviceId>
    <xeDeviceVersion>1</xeDeviceVersion>
    <xaAttributeType>37</xaAttributeType>
    <xadAttributeStringValue>5011</xadAttributeStringValue>
    <xadAttributeIntegerValue>5011</xadAttributeIntegerValue>
    <xadUpdateTime>2010-06-14T16:07:43Z</xadUpdateTime>
  </XbeeAttributeDataHistoryCore>
  <XbeeAttributeDataHistoryCore>
    <id>
      <xpExtAddr>00:13:A2:00:40:66:33:03</xpExtAddr>
      <xeEndpointId>1</xeEndpointId>
      <xcClusterType>0</xcClusterType>
      <xcClusterId>1794</xcClusterId>
      <xaAttributeId>0</xaAttributeId>
      <xadhId>687</xadhId>
    </id>
    <csId>3</csId>
    <devConnectwareId>00000000-00000000-00000000-00000000</devConnectwareId>
    <xpParentAddr>65534</xpParentAddr>
    <xeProfileId>265</xeProfileId>
    <xeDeviceId>1280</xeDeviceId>
    <xeDeviceVersion>1</xeDeviceVersion>
    <xaAttributeType>37</xaAttributeType>
    <xadAttributeStringValue>5017</xadAttributeStringValue>
    <xadAttributeIntegerValue>5017</xadAttributeIntegerValue>
    <xadUpdateTime>2010-06-14T16:21:43Z</xadUpdateTime>
  </XbeeAttributeDataHistoryCore>
</result>
```

9.8 XbeeAttributeDataHistoryFull

URI: <http://<hostname>/ws/ XbeeAttributeDataHistoryFull>

HTTP operations supported:

- GET: Display a list of the historical ZigBee attribute values - the Full variation contains more data than the Core
- DELETE

9.9 XbeeAttributeReportingCore (deprecated 2.9.0)

As of 2.9.0 this API is **deprecated** but is still available for compatibility. If your devices run Smart Energy Gateway 1.6 or later you should use the *start_reports* and *get_local_reporting_configurations* commands (via SCI) to configure reporting and query the reporting configurations. You should **not** use *XbeeAttributeReportingCore* API in conjunction with *start_reports*. They are not compatible and results may be unpredictable.

URI: <http://<hostname>/ws/ XbeeAttributeReportingCore>

HTTP operations supported:

- GET: Display a list of the most recent ZigBee attribute reporting configuration values
- PUT: Creates or updates the attribute reporting configuration
- DELETE: Terminate the attribute reporting configuration

This [API](#) is used to read and update the attribute reporting configuration.

Elements include:

- csId - Device Cloud ID of customer owning the gateway that discovered this attributes node
- devConnectwareId - Device ID of the gateway that discovered this attributes node
- xpExtAddr - ZigBee 64bit extended address of node
- xeEndpointId - ZigBee endpoint this attribute resides on
- xeProfileId - associated ZigBee profile

- xcClusterId - associated ZigBee cluster
- xcClusterType - kind of associated ZigBee cluster (0=server or 1=client)
- xaAttributeId - ZigBee attribute ID
- xarMinReportingInterval - minimum interval, in seconds, between issuing reports of the attribute
- xarMaxReportingInterval - maximum interval, in seconds, between issuing reports of the attribute
- xarReportableChange - the minimum change to the attribute that will result in a report being issued. Applicable for non discrete attributes. Value type matches attribute.
- xarTimeout - the maximum expected time, in seconds, between received reports for the attribute - else consider problem
- xarEnabled - indicates if this reporting configuration record is enabled or not
- xarUpdateTime - time that the reporting configuration is reported from the device
- xarLocalEndpointId - 8-bit identifier of the local endpoint
- xarReportingDirection - the cluster on the local device that is transmitting(0) or receiving(1) reports
- xarPseudoReporting - when set to TRUE, reporting is using the Smart Energy pseudo reporting mechanism
- xarCheckInterval - when xarPseudoReporting is enabled, this is the interval between read attribute requests, in seconds

The following example will enable reports for the Current Summation Delivered attribute on the Metering server on device 11:22:33:44:55:66:77:88, endpoint 16:

PUT ws/XbeeAttributeReportingCore

body:

```
<XbeeAttributeReportingCore>
  <id>
    <xpExtAddr>11:22:33:44:55:66:77:88</xpExtAddr>
    <xeEndpointId>16</xeEndpointId>
    <xcClusterType>0</xcClusterType>
    <xcClusterId>1794</xcClusterId>
    <xaAttributeId>0</xaAttributeId>
  </id>
  <devConnectwareId>00000000-00000000-00000000-00000000</devConnectwareId>
  <xarMinReportingInterval>60</xarMinReportingInterval>
  <xarMaxReportingInterval>180</xarMaxReportingInterval>
  <xarReportableChange>1</xarReportableChange>
  <xarTimeout>120</xarTimeout>
  <xarEnabled>true</xarEnabled>
</XbeeAttributeReportingCore>
```

9.10 XbeeClusterCore

This API is used by clients to identify one or more clusters of any node in their set of HAN networks. Similar to the concept of XbeeAttributeCore, this interface is useful if you need to list only the clusters that are available on any or all nodes that you have access to. Since this interface does not include attributes or attribute data, it is a more efficient interface to use if all you need is a list of clusters.

GET ws/XbeeClusterCore[?param1¶m2...¶mn]

This API is used to query portions of the available HAN network topology. Callers can scope the returned information through the use of condition parameters.

- devConnectwareId - Connectware ID of the gateway discovering this cluster's node
- xpExtAddr - Zigbee 64bit extended address of node
- xpParentAddr - If node is an endnode this will be the ext addr of the router it connects to. If node is a router this will be 0xFFFF.
- xeEndpointId - Zigbee endpoint this cluster resides on
- xeProfileId - associated Zigbee profile
- xeDeviceId - associated Zigbee device type
- xeDeviceVersion - associated Zigbee device version
- xcClusterId - associated Zigbee cluster
- xcClusterType - kind of associated Zigbee cluster (0=server or 1=client)

9.11 XbeeEventDataCore

This web service is used to query ZCL events (commands) received by the ESI or Aux Gateway of the HAN network. As these events are reported, they are timestamped and saved in the server for later retrieval through the EventData interface. Currently supported events include pricing, DRLC, and messages.

To query current event values use the XbeeEventDataCore Full controller. To query historical readings use the XbeeEventDataHistoryCore Full controller.

URI: http://<hostname>/ws/ XbeeEventDataCore

HTTP operations supported:

- GET: Display a list of the most recent ZigBee event data.
- DELETE

Example:

This API is used to query reported events such as DRLC, Pricing, or Message events. Events are generally identified by the endpoint, clusterId, and eventId. The commandId indicates the last command sent for the event. The xedPayload contains the actual event record received. The eventId will match the event-specific identifier (issuer_event_id for pricing and DRLC, and message_id for messages).

The following example reads all the events received by a particular gateway. Note that values usually referred to in a hex notation must be converted to integer values (0x109 = 265, 0x700 = 1792).

GET ws/XbeeEventDataHistoryCore?condition=devConnectwareId='00000000-00000000-00409DFF-FF3D7115'

returns:

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <resultTotalRows>3</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>3</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <XbeeEventDataCore>
    <id>
      <xpExtAddr>00:13:A2:00:40:5C:0A:45</xpExtAddr>
      <xeEndpointId>94</xeEndpointId>
      <xcClusterType>1</xcClusterType>
      <xcClusterId>1792</xcClusterId>
      <xedEventId>13124</xedEventId>
    </id>
    <csId>3</csId>
    <devConnectwareId>00000000-00000000-00000000-00000000</devConnectwareId>
    <xeProfileId>265</xeProfileId>
    <xeDeviceId>1282</xeDeviceId>
    <xeDeviceVersion>1</xeDeviceVersion>
    <xedCommandId>0</xedCommandId>
    <xedPayload>
      <record type="PublishPriceRecord">
        <current_time type="int">0x140EC5BF</current_time>
```

```

<provider_id type="int">0xFFFFFFFF</provider_id>
<duration_in_minutes type="int">0x2D0</duration_in_minutes>
<unit_of_measure type="int">0x0</unit_of_measure>
<price_trailing_digit_and_price_tier type="int">0x33</price_trailing_digit_and_price_tier>
<start_time type="int">0x0</start_time>
<number_of_price_tiers_and_register_tier type="int">0x43</number_of_price_tiers_and_register_tier>
<alternate_cost_delivered type="int">0xFFFFFFFF</alternate_cost_delivered>
<currency type="int">0x348</currency>
<issuer_event_id type="int">0x3344</issuer_event_id>
<alternate_cost_unit type="int">0x1</alternate_cost_unit>
<alternate_cost_trailing_digit type="int">0xFF</alternate_cost_trailing_digit>
<price_ratio type="int">0xFF</price_ratio>
<generation_price type="int">0xFFFFFFFF</generation_price>
<price type="int">0xA</price>
<generation_price_ratio type="int">0xFF</generation_price_ratio>
<rate_label type="string">SamplePrice</rate_label>
</record>
</xEdPayload>
<xEdStartTime>2010-08-30T19:40:45Z</xEdStartTime>
<xEdEndTime>2010-08-31T07:40:45Z</xEdEndTime>
<xEdActive>false</xEdActive>
<xEdUpdateTime>2010-08-30T22:09:39Z</xEdUpdateTime>
</XbeeEventDataCore>
<XbeeEventDataCore>
  <id>
    <xpExtAddr>00:13:A2:00:40:5C:0A:45</xpExtAddr>
    <xeEndpointId>94</xeEndpointId>
    <xcClusterType>1</xcClusterType>
    <xcClusterId>1793</xcClusterId>
    <xEdEventId>26436</xEdEventId>
  </id>
  <csId>3</csId>
  <devConnectwareId>00000000-00000000-00000000-00000000</devConnectwareId>
  <xeProfileId>265</xeProfileId>
  <xeDeviceId>1282</xeDeviceId>
  <xeDeviceVersion>1</xeDeviceVersion>
  <xEdCommandId>0</xEdCommandId>
  <xEdPayload>
    <record type="LoadControlEventRecord">
      <duration_in_minutes type="int">0x258</duration_in_minutes>
      <cooling_temperature_offset type="int">0xFF</cooling_temperature_offset>
      <event_control type="int">0x3</event_control>
      <start_time type="int">0x0</start_time>
      <device_class type="int">0xFF</device_class>
      <cooling_temperature_set_point type="int">-0x8000</cooling_temperature_set_point>
      <heating_temperature_set_point type="int">-0x8000</heating_temperature_set_point>
      <issuer_event_id type="int">0x6744</issuer_event_id>
      <duty_cycle type="int">0xFF</duty_cycle>
      <average_load_adjustment_percentage type="int">-0x80</average_load_adjustment_percentage>
      <heating_temperature_offset type="int">0xFF</heating_temperature_offset>
      <criticality_level type="int">0x1</criticality_level>
      <utility_enrollment_group type="int">0x0</utility_enrollment_group>
    </record>
  </xEdPayload>
  <xEdStartTime>2010-08-30T22:09:03Z</xEdStartTime>
  <xEdEndTime>2010-08-31T08:09:03Z</xEdEndTime>
  <xEdActive>false</xEdActive>
  <xEdUpdateTime>2010-08-31T08:09:03Z</xEdUpdateTime>
</XbeeEventDataCore>
<XbeeEventDataCore>
  <id>
    <xpExtAddr>00:13:A2:00:40:5C:0A:45</xpExtAddr>
    <xeEndpointId>94</xeEndpointId>
    <xcClusterType>1</xcClusterType>
    <xcClusterId>1795</xcClusterId>
    <xEdEventId>26231</xEdEventId>
  </id>
  <csId>3</csId>
  <devConnectwareId>00000000-00000000-00000000-00000000</devConnectwareId>
  <xeProfileId>265</xeProfileId>
  <xeDeviceId>1282</xeDeviceId>
  <xeDeviceVersion>1</xeDeviceVersion>
  <xEdCommandId>0</xEdCommandId>
  <xEdPayload>
    <record type="DisplayMessageRecord">
      <message_control type="int">0x80</message_control>
      <start_time type="int">0x0</start_time>
      <duration_in_minutes type="int">0x258</duration_in_minutes>
      <message type="string">Hello world!</message>
      <message_id type="int">0x6677</message_id>
    </record>
  </xEdPayload>
  <xEdStartTime>2010-08-30T19:42:32Z</xEdStartTime>
  <xEdEndTime>2010-08-31T05:42:32Z</xEdEndTime>
  <xEdActive>false</xEdActive>
  <xEdUpdateTime>2010-08-30T22:07:51Z</xEdUpdateTime>
</XbeeEventDataCore>
</result>

```

9.12 XbeeEventDataFull

URI: <http://<hostname>/ws/ XbeeEventDataFull>

HTTP operations supported:

- GET: Display a list of the most recent ZigBee event data - the Full variation contains more data than the Core
- DELETE

9.13 XbeeEventDataHistoryCore

URI: <http://<hostname>/ws/ XbeeEventDataHistoryCore>

HTTP operations supported:

- GET: Display a list of the historical ZigBee event data
- DELETE: Delete historical data

9.14 XbeeEventDataHistoryFull

URI: <http://<hostname>/ws/ XbeeEventDataHistoryFull>

HTTP operations supported:

- GET: Display a list of the historical ZigBee event data - the Full variation contains more data than the Core
- DELETE: Delete historical data

10. DIA (Device Integration Application)

10.1 Introduction

Customers can build custom remote sampling solutions that report data through DIA. The data will be stored in the database's DIA tables instead of being stored as files. This will allow Device Cloud users to query DIA tables directly. You can enable this feature using the DIA data service subscription.

The SMS presentation data pushed to Device Cloud will also be stored in these tables.

Callers can specify an aggregate constraint to perform special operations against the matching result set. The aggregate operations are: SUM, AVG, MAX, MIN, COUNT. See the examples below for more information.

NOTE: For this release, only the idigi_db presentation data is stored in the tables. The RCI presentation data queried via Device Cloud is not stored in these tables.

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, Device Cloud may not be able to parse the sample data uploaded from the device. If the idigi_db presentation cannot be parsed by Device Cloud, the sample data won't be stored in the DIA tables; instead it will default to being stored as a file.

10.2 Aggregate Operations

Callers can specify an aggregate constraint to perform special operations against the matching result set. The aggregate operations are: SUM, AVG, MAX, MIN, COUNT.

Example:

To return the average value of the samples for a DIA channel, the query would be:

GET ws/DiaChannelDataHistoryFull?condition=devConnectwareId='00000000-00000000-00409DFF-FF32E1F7' and dcChannelName='counter'&aggregate=AVG

Which returns:

```
<result>
  <resultTotalRows>1</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>1</resultSize>
  <requestedSize>1</requestedSize>
  <remainingSize>0</remainingSize>
  <DiaChannelDataHistoryFull>
    <avg>47.0</avg>
  </DiaChannelDataHistoryFull>
</result>
```

10.3 DiaChannelDataFull

URI: <http://<hostname>/ws/ DiaChannelDataFull>

HTTP operations supported:

- GET: Display a list of the most recent DIA channel data values
- DELETE

Example:

Either of the following will delete the specified DIA channel (these are equivalent requests):

DELETE ws/DiaChannelDataFull/00000000-00000000-00409DFF-FF32E1F7/template/counter

Or

DELETE ws/DiaChannelDataFull?condition=devConnectwareId='00000000-00000000-00409DFF-FF32E1F7' and ddInstanceName='template' and dcChannelName='counter'

10.4 DiaChannelDataHistoryFull

URI: <http://<hostname>/ws/ DiaChannelDataHistoryFull>

HTTP operations supported:

- GET: Display a list of the historical DIA channel data values
- DELETE

Elements include:

- cstId - Device Cloud ID of customer owning the gateway
- devConnectwareId - Device ID of the gateway
- xpExtAddr - ZigBee 64bit extended address of node (**deprecated 2.9.0**)
- ddInstanceName - DIA device's instance name
- dcChannelName - DIA channel name
- dcDataType - the type of data element returned (e.g. float, integer, string)
- dcUnits - The reported value's unit of measure (e.g. meters, Fahrenheit, Volts). This is the value element in the DIA channel sample.
- dcdUpdateTime - The time when the driver reports that the value was acquired. This is the timestamp as recorded by the device in the DIA channel sample.
- dcdStringValue - string form of the value
- dcdIntegerValue - integer value (only available if value format is integer, else null)
- dcdFloatValue - float value (only available if value format is float, else null)
- dcdDateValue - date value (only available if value is a date, else null)
- dcdBooleanValue - boolean value (only available if value is boolean, else null)
- dcdHid - unique numeric ID assigned by Device Cloud for historical records only

As of 2.9.0 the *xpExtAddr* element is **deprecated** but is still available for compatibility.

Examples:

Example #1

GET ws/DiaChannelDataFull?condition=devConnectwareId='00000000-00000000-00409DFF-FF32E1F7'

Which returns:

```
<result>
  <resultTotalRows>1</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>1</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <DiaChannelDataFull>
    <id>
      <devConnectwareId>00000000-00000000-00409DFF-FF32E1F7</devConnectwareId>
      <ddInstanceName>template</ddInstanceName>
      <dcChannelName>counter</dcChannelName>
    </id>
    <dcdUpdateTime>2011-01-17T16:35:21Z</dcdUpdateTime>
    <dcdStringValue>248650</dcdStringValue>
    <dcdIntegerValue>248650</dcdIntegerValue>
  </DiaChannelDataFull>
</result>
```

This represents the following DIAidigi_db presentation data pushed from the device with ID '00000000-00000000-00409DFF-FF32E1F7':

```
<idigi_data>
  <sample>
    <name>template.counter</name>
    <value>248650</value>
    <unit />
    <timestamp>2011-01-17T16:35:21Z</timestamp>
  </sample>
</idigi_data>
```

Example #2

To query the historical data for samples outside of a range:

GET ws/DiaChannelDataHistoryFull?condition=(dcdIntegerValue>120000 or dcdIntegerValue<1000)

Which returns:

```
<result>
  <resultTotalRows>635</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>635</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <DiaChannelDataHistoryFull>
    <id>
      <devConnectwareId>00000000-00000000-00409DFF-FF32E1F7</devConnectwareId>
      <ddInstanceName>template</ddInstanceName>
      <dcChannelName>counter</dcChannelName>
      <dcdhId>20502</dcdhId>
    </id>
    <cstId>12</cstId>
    <dcDataType>0</dcDataType>
    <dcdUpdateTime>2011-02-03T09:53:57Z</dcdUpdateTime>
    <dcdStringValue>1200157</dcdStringValue>
    <dcdIntegerValue>1200157</dcdIntegerValue>
  </DiaChannelDataHistoryFull>
  <DiaChannelDataHistoryFull>
    <id>
      <devConnectwareId>00000000-00000000-00409DFF-FF32E1F7</devConnectwareId>
      <ddInstanceName>template</ddInstanceName>
      <dcChannelName>counter</dcChannelName>
      <dcdhId>20506</dcdhId>
    </id>
    <cstId>12</cstId>
    <dcDataType>0</dcDataType>
    <dcdUpdateTime>2011-02-03T09:58:00Z</dcdUpdateTime>
    <dcdStringValue>1200392</dcdStringValue>
    <dcdIntegerValue>1200392</dcdIntegerValue>
  </DiaChannelDataHistoryFull>
  .... // additional results were truncated for this doc
</result>
```

Example #3

Either of the following will delete the historical data samples for the specified DIA channel (these are equivalent requests):

DELETE ws/DiaChannelDataHistoryFull/00000000-00000000-00409DFF-FF32E1F7/template/counter

Or

DELETE ws/DiaChannelDataHistoryFull?condition=devConnectwareId='00000000-00000000-00409DFF-FF32E1F7' and ddInstanceName='template' and dcChannelName='counter'

Example #4

To delete the historical data samples that occurred earlier than a given time:

DELETE ws/DiaChannelDataHistoryFull?condition=dcdUpdateTime<'2012-02-29T00:00:00Z'

11. SMS

11.1 Receiving SMS Messages

There are two types of SMS data:

1. **Data** - Data SMS messages are sent from the device using the python function `idigisms.send()` and are stored in `FileData`. Data from the device is stored as a file in storage.
 - o Python programs specify the data contents and, optionally, the file name (called 'path' in `idigisms.send()`).
 - o If a path is not specified, the file will be stored with a name of `SmsUnnamed` and will be archived (`fdArchive` true).
2. **DIA Data** - Device Cloud will parse DIA messages and add them to the DIA data specific storage or as a file to the generic storage depending on whether the device has the DIA Data Management service subscription or not.

11.2 Sending SMS Messages via Web Services

Device Cloud sends SMS requests to registered devices via **SCI**. Device Cloud SMS 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 Device Cloud to create jobs. These jobs can be synchronous or asynchronous and job results are retrieved the same way they are for **RCI** jobs.

`<send_message>` options have the following effect with Device Cloud SMS:

- `synchronous="true|false, syncTimeout"`

Behavior is identical to existing **SCI** requests.

- `reply="all|errors|none"`

This 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 Device Cloud SMS 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"`

Device Cloud does not currently service SMS 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".

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

11.3 SMS Command Children

Command	Description
ping	Request to determine whether device is able to receive SMS messages - No parameters
provision	Set a Device Cloud phone number and optionally the service ID in the device. The "restrict sender" flag must be off in the device for this command. - No parameters
raw	Sends a raw SMS message from a device with no modification from Device Cloud. This is useful in cases when customers wish to use every byte of the SMS message (Device Cloud protocol takes approximately 5 bytes per message of overhead), or when using a device that doesn't have Device Cloud 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 device immediately - No parameters
request_connect	Request an Device Cloud 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. If the response is larger than this value, it will be truncated. The value is the number of SMS 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 <code>do_command</code> 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.
dia	Send a DIA command to the running DIA instance. The format of this request is documented in the DIA SMS presentation documentation.
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 <code>format="base64 text"</code> 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 <code>format="base64"</code> attribute. That is, even if a request uses <code>format="text"</code> , the reply may have <code>format="base64"</code> set.

11.4 Request Connect - SMS Support

Device Cloud can be used to send an SMS "request connect" message to a Device Cloud-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 Device Cloud 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 Device Manager as well.

11.4.1 Configure Device Cloud with the Phone Number of the Device

Device Cloud needs to be informed of the device's phone number. When a registered device connects for the first time, the phone number is read from it and automatically provisioned. There are cases where auto provisioning will not work (cellular is not configured yet, the device is provisioned without it ever being connected, etc). A manual provisioning method solves the problem.

To provision the phone number of a Device Cloud-registered device, the phone number is added to an entry in `NetworkInterface` that represents a SIM installed in a device.

1.Retrieve phone number from Device Cloud-registered device

The phone number of the device can be retrieved using the following **RCI** request and example response (the phone number is indicated below):

```
<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>
    </mobile_stats>
  </query_state>
</rci_reply>
```

```

<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>S: 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>

```

2. Find existing NetworkInterface record to update

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 a nild value, skip this step and proceed to step 4.

To find the NetworkInterface record to update for your Device Cloud-registered device:

Perform a GET on /ws/DeviceInterface/?condition=devConnectwareId='00000000-00000000-00409DFF-FF2EB94D' (replace with your device's Device ID).

An example 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>
    <csiId>10</csiId>
    <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>
  </DeviceInterface>
</result>

```

```

<niTerminated>false</niTerminated>
<niPhone>N/A</niPhone>
<niActivePhone>false</niActivePhone>
<niIdigiPhone>32075</niIdigiPhone>
</DeviceInterface>
</result>

```

Find the nild of the NetworkInterface record to be updated; nild is 26 in the above example. Retrieve the NetworkInterface record using the ID found above:

aws/NetworkInterface/26/0 (replace with your device's nild, 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>

```

3. Update NetworkInterface Record

To update the NetworkInterface record with the device's Modem ID, copy the contents of <NetworkInterface> from the GET above. Update the niPhone tag with the phone number you discovered in step 1 (replace N/A with your device's 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 Device Cloud 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 /aws/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>

```

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).

a. First, you must find the the devId for your Device Cloud-registered device (indicated in the example below):

Perform a GET on /aws/DeviceCore?condition=devConnectwareId=00000000-00000000-00409DFF-FF4A3946" (replace with your device's 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>
    <dpCapabilities>2579</dpCapabilities>
  </DeviceCore>

```

</result>

- b. 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.
- c. Ensure the niInterfaceType value is set to 0.
- d. 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/sci

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

11.4.2 Configure Device to Receive SMS Commands

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

RCI request:

```
<rci_request version="1.1">
  <set_setting>
    <smcell>
      <state>on</state>
    </smcell>
    <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://login.etherios.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>
```

11.4.3 RCI for SMS

RCI group: idigisms

Field	Options	Description
state	on, off	Device Cloud 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 Device Cloud. 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 Device Cloud's server. This needs to be obtained from Digi (each cluster has its own phone number).
restrict_sender	on, off	If on, only Device Cloud SMS messages originating from "phnum" and with the service ID "service_identifier" will be honored as Device Cloud SMS messages.

RCI group: smscell

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

RCI group: mgmtconnection index = "1" (client initiated Device Cloud connections)

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

RCI group: mgmtconnection index = "4" (paged - i.e. temporary - - Device Cloud connections)

Field	Options	Description
connectionEnabled	on, off	Enables temporary connections. A connection request results in a paged Device Cloud 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 Device Cloud's 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 Device Cloud 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.

11.4.4 Send Device Cloud 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 Device Cloud-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 Device Cloud. 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 Device Cloud 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

11.4.5 Wait for Device to Connect

The connection status of any Device Cloud-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 the id is the id associated with a particular device.

11.4.6 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>
```

12. Satellite

12.1 Sending and Receiving Iridium Satellite Messages

Device Cloud sends Iridium requests to Device Cloud-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 Device Cloud 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:

- synchronous="true|false", syncTimeout

Behavior is identical to existing SCI requests.

- reply="all|errors|none"

This 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"

Device Cloud does not currently service Iridium 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".

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

12.2 Iridium Satellite Command Children

Command	Description
ping	Request to determine whether device is able to receive Iridium satellite messages - No parameters
raw	Send a raw Iridium satellite message from a device with no modification from Device Cloud; 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 (Device Cloud protocol takes approximately 5 bytes per message of overhead), or when using a device that doesn't have Device Cloud 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 Device Cloud 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 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.

12.3 Request Connect - Iridium Support

Device Cloud 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 Device Cloud 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 Device Cloud's UI as well.

12.3.1 Configure Device Cloud with the Modem ID of the Device

Device Cloud needs to be informed of the device's 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, etc). A manual provisioning method solves this problem.

To provision the Modem ID of a device in Device Cloud, 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.

1.Retrieve Modem ID from Device Cloud-registered device

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

```
<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 this number. Later in this example you will need to use this number, based off of your device's Device ID, in order to update your configuration. Copy only the actual Modem ID number (excluding the <serial_number> tags).

```
<?xml version="1.0" encoding="UTF-8"?>
<sci_reply version="1.0">
  <send_message>
    <device id="000000000000-00000000-0004F3FF-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>

```

2. Find existing NetworkInterface record to update

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 nild value, skip this step and proceed to step 4.

To find the NetworkInterface record to update for your Device Cloud-registered device:

Perform a GET on `/ws/DeviceInterface/?condition=devConnectwareId='00000000-00000000-0004F3FF-FF03A8A5'` (replace with your device's Device ID).

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>
    <csId>2</csId>
    <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 nild of the NetworkInterface record to be updated; nild is 133 in the above example. Retrieve the NetworkInterface record using the nild number found above:

`/ws/NetworkInterface/133/0` (replace with your device's nild, 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>
    <csId>2</csId>
    <grpId>2</grpId>
    <niEffectiveStartDate>2011-12-15T00:05:00.000Z</niEffectiveStartDate>
    <niTerminated>false</niTerminated>
    <niActivePhone>false</niActivePhone>
  </NetworkInterface>
</result>

```

3. Update NetworkInterface Record

To update the NetworkInterface record with the device's 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's 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>
  <csId>2</csId>
  <grpId>2</grpId>
  <niEffectiveStartDate>2011-12-15T00:05:00.000Z</niEffectiveStartDate>
  <niTerminated>false</niTerminated>

```

```
<niActivePhone>false</niActivePhone>
</NetworkInterface>
```

4. Configure Modem ID 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).

a. First, you must find the the devId for your Device Cloud-registered device (indicated in the example below):

Perform a GET on /ws/DeviceCore?condition=devConnectwareId='00000000-00000000-0004F3FF-FF03A80C' (replace with your device's 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>
    <csiId>2</csiId>
    <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>
    <dpCapabilities>819</dpCapabilities>
    <dpTags>,techpubs,</dpTags>
  </DeviceCore>
</result>
```

b. 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.

c. 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>
```

12.3.2 Configure Device to Receive Iridium Satellite Commands

The following example RCI request will configure a Device Cloud-registered device to enable Iridium Satellite Support, configure Iridium Satellite Support, disable client initiated Device Cloud connections, and configure paged Device Cloud 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://login.etherios.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>
```

```
</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).

12.3.3 Send an Iridium Satellite Request Connect

To send a connect request to a Device Cloud-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>
```

For more information on checking the status of the request connect command, see the [Status for a Particular Job](#) section.

Details:

[SCI](#) is used to send Iridium requests to Device Cloud-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 Device Cloud. 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

12.3.4 Wait for Device to Connect

The connection status of any Device Cloud-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 the id is the id associated with a particular device.

12.3.5 Send a Disconnect

Once work is complete to a device, a web services client may optionally disconnect the device from Device Cloud:

POST /ws/sci

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

13. SM/UDP

13.1 Introduction

The SMUDP (Short Message/User Datagram Protocol) feature allows devices to leverage the very small data footprint of Device Cloud SM protocol (currently used for SMS and Iridium Satellite messaging) over UDP. However, it is important to note that SMUDP requests vary greatly from SMS/Iridium requests as SMUDP 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 Device Cloud 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 Device Cloud.

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 SMUDP 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 SMUDP request (known as a datagram) to Device Cloud, Device Cloud 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), Device Cloud will check for queued messages and send them down to the device. For example, if you send the SMUDP Reboot request to your device, the device will not reboot immediately. Instead, the SMUDP Reboot request will be queued for the device. The next time an SMUDP request is sent to the device, Device Cloud will check for queued messages and send the queued SMUDP 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).

To learn more about the SMUDP feature and for troubleshooting information, see the "SMUDP" Chapter of the *Device Cloud User's Guide* (available within the Documentation tab).

13.2 Sending and Receiving SM/UDP messages via Web Services

Device Cloud sends SMUDP requests to Device Cloud-registered devices via [SCI](#). SMUDP 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 Device Cloud 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 SMUDP 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.

NOTE: See the [Retrieve Status](#) section for more information about retrieving the status of an asynchronous request.

<send_message> options have the following effect with SMUDP:

- synchronous="true|false, syncTimeout"

Behavior is identical to existing SCI requests.

- reply="all|errors|none"

This 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 SMUDP 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"

Device Cloud does not currently service SMUDP 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".

SMUDP requests are specified by the tag <sm_udp> as a child element of <send_message>.

13.3 SMUDP 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 Device Cloud 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. 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.

13.4 Request Connect - SM/UDP Support

Device Cloud can be used to send a SMUDP "request connect" message to a Device Cloud-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 Device Cloud 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 Device Cloud's UI as well.

13.4.1 Configure Device to Receive SM/UDP Commands

The following example will configure a Device Cloud-registered device to enable SMUDP.

POST to /ws/DeviceCore to provision a device with SMUDP 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 SMUDP.

In the event that you'd like to disable SMUDP for a device, use the following example.

PUT to /ws/DeviceCore to update the device and disable SMUDP.

```
<DeviceCore>
  <devConnectwareId>00000000-00000000-00409DFF-FF000000</devConnectwareId>
  <dpUdpSmEnabled>false</dpUdpSmEnabled>
</DeviceCore>
```

After sending this request the device will no longer be configured for SMUDP.

13.4.2 Send SM/UDP Request Connect

To send a connect request to a device via SMUDP, 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>
```

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

Details:

SCI is used to send SMUDP requests to Device Cloud-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 Device Cloud. These jobs can be synchronous or asynchronous and job results are retrieved the same way they are for RCI jobs.

NOTE: It is suggested SMUDP requests be done asynchronously as they can take a while and requests done synchronously may time out before the response has been received.

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

- synchronous="true|false"

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

SMUDP 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

13.4.3 Wait for Device to Connect

The connection status of any Device Cloud-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.

13.4.4 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>
```

14. Monitor (Push Notifications)

14.1 Introduction

The Web Services Monitor API enables customer applications to register for asynchronous (push) notification of various events within Device Cloud. The API supports setting up, modifying, and deleting event monitors. Each monitor can be configured to report on one or more of the various data or status Device Cloud events. For example, if you are monitoring DeviceCore and a new device was added to your Device Cloud account, then Device Cloud would push a DeviceCore CREATE message that included that information about the new device. This feature currently supports both TCP and HTTP connections between the monitoring application and Device Cloud.

To monitor Device Cloud activity, the registering application chooses which types of events they are interested in and what type of notification mechanism to use. As events of interest occur within Device Cloud, it will push to the application via the selected mode of transport. See the [Transport Protocols for the Monitor API](#) Appendix for more information.

The event activities to be monitored may include:

- DataReports: These events include data pushed into Device Cloud from remote gateways in the form of FileData, XbeeAttributeData, DiaChannelData, etc.
- StatusReports: These events include general status updates for things like gateway connection status, remote device availability, etc.
- AlarmReports: These events include alarm status updates when alarms are triggered or reset.

14.2 The Monitor Web Services Request

Monitor is used by clients to register for push notification. This feature is available with a Device Cloud subscription to the Push Monitor service. Since the monitor ID is automatically assigned with the POST request, this element of the monitor cannot be changed. Each additional monitor added will be given a unique ID.

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

HTTP operations supported:

- GET: Retrieves a list of the configured monitors
- POST: Create new monitor
- PUT: Update monitor
- DELETE: Delete monitor

Elements include:

- monId - the generated ID for the monitor
- cstId - Device Cloud ID of customer
- monFormatType - Specifies the format of the delivered event data. The options are 'xml' or 'json'.
- monTopic - Specifies the topics to be monitored. A topic is specified by the resource name followed optionally by the resource ID using the normal REST slash convention. Example topics are:

"XbeeAttributeDataCore" which matches all XbeeAttributeData events reported under this user.

"XbeeAttributeDataCore/00:13:A2:00:40:00:00:00" which matches all XbeeAttributeData events reported by the node with xpExtAddr 00:13:A2:00:40:00:00:00. Multiple comma-separated topics can be specified.

By default, monitoring a topic will return all types of operations against that topic including creates, updates, and deletes. You can further scope your interest to certain operations by prepending an optional operation qualifier to the beginning of the topic string. For example:

"[operation=U]XbeeAttributeDataCore" matches all XbeeAttributeDataCore update operations.

"[operation=C,D]DeviceCore" matches all DeviceCore create and delete operations.

NOTE: In the above examples 'C' denotes a create operation, 'D' denotes a delete operation, and 'U' denotes an update operation.

Resources can be scoped to a group name by adding a special component in the topic string. This is specified as follows:

"[group=America,Columbia]DeviceCore", matches DeviceCore operations for devices which are assigned to the America or Columbia group.

NOTE: The order to which you prepend the group and operation qualifiers does not matter.

Regular expressions can be specified in the ID portion of a topic by starting the ID portion with the character '*' followed by the regular expression. For example:

"XbeeCore/FE" matches all the XbeeCore nodes who's extended address starts with 'FE'.

"XbeeCore/[02468ACE]" matches all XbeeCore nodes who's extended address ends in an even number.

- monTransportType - Specifies the transport method to be used to deliver push modifications to a customer application. Currently, 'tcp' and 'http' are the only supported transport types.

NOTE: See the [Transport Protocols for the Monitor API](#) Appendix for more information.

- monBatchSize - specifies an upper bound on how many messages will be aggregated together before sending them. The maximum allowed value is 1000. The default is 1.
- monBatchDuration - specifies an upper bound on how many seconds messages will be aggregated before sending them. the default is 0.
- monCompression - Specifies what method should be used to compress the messages. Options are 'zlib' or 'none'; the default is 'none'. If 'zlib' is specified, the deflate algorithm is used to compress the data therefore, you should correspondingly use inflate to decompress the data.

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

- monLastConnect - returned in the GET response, specifies the time the last connection was established to the customer's application.
- monLastSent - returned in the GET response, specifies the time the last message was pushed to the customer's application.
- monStatus - returned in the GET response, specifies the current status of the connection to the customer's application. Possible values are:
 - ACTIVE - monitor is connected and publishing events
 - INACTIVE - monitor is not connected; events are not published or recorded
 - SUSPENDED - monitor is not connected; events are being recorded for later replay
 - DISABLED - monitor requires reconfiguration
 - CONNECTING - monitor is attempting to connect to the customer's server
 - monTransportUrl - http transport only, specifies the URL of the customer's web server. The url should be of the form: http[s]://customer.domain.com/application/path.
- monTransportToken - http only, the token specifies the credentials for basic authentication, in the format of 'username:password'.
- monAutoReplayOnConnect - when enabled, the monitor will resend any events that may have been missed due to network or server outages. When disabled, these missed event will simply be discarded. The default is false.
- monDescription - this is an optional text field which can be used to label or describe the monitor.

14.2.1 Example 1 - Creating a Basic Monitor

The following example describes how to create a new monitor using two topics: DeviceCore and XbeeAttributeDataCore. In this example you will monitor all operations pertaining to a single device.

NOTE: The DeviceCore and XbeeAttributeDataCore topics are being used for demonstration purposes only. You can follow this example using any of the supported monitor topics.

14.2.1.1 Create new TCP monitor using POST

First, use a POST to set up the monitor:

POST /ws/Monitor

```
<Monitor>
  <monTopic>DeviceCore,XbeeAttributeDataCore/00:13:A2:00:40:00:00:00</monTopic>
  <monTransportType>tcp</monTransportType>
  <monFormatType>xml</monFormatType>
  <monBatchSize>1</monBatchSize>
  <monCompression>none</monCompression>
  <monBatchDuration>0</monBatchDuration>
</Monitor>
```

Which will return the following response:

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <location>Monitor/267</location>
</result>
```

Notice that the response contains the monitor ID of 267.

14.2.1.2 Create new HTTP monitor using POST

To create a monitor that uses the HTTP transport:

POST /ws/Monitor

```
<Monitor>
  <monTopic>DeviceCore,XbeeCore</monTopic>
  <monTransportType>http</monTransportType>
  <monTransportUrl>your website url</monTransportUrl>
  <monTransportToken>username:password</monTransportToken>
  <monFormatType>json</monFormatType>
  <monBatchSize>1</monBatchSize>
  <monCompression>none</monCompression>
  <monBatchDuration>0</monBatchDuration>
</Monitor>
```

14.2.1.3 List configured monitors using GET

Now use a GET to see this monitor in place (the following is the response from the example GET). You can see that monitor 267 is in place:

GET /ws/Monitor

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <resultTotalRows>1</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>1</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <Monitor>
    <monId>267</monId>
    <csId>27</csId>
    <monTopic>DeviceCore,XbeeAttributeDataCore/00:13:A2:00:40:00:00:00</monTopic>
    <monTransportType>tcp</monTransportType>
    <monFormatType>xml</monFormatType>
    <monBatchSize>1</monBatchSize>
    <monCompression>none</monCompression>
    <monStatus>INACTIVE</monStatus>
```

```
<monBatchDuration>0</monBatchDuration>
</Monitor>
</result>
```

14.2.1.4 Modify an existing monitor using PUT

Next, perform an update using a monitor PUT. In this case, changing the format type from 'xml' to 'json'.

PUT /ws/Monitor

```
<Monitor>
  <monId>267</monId>
  <monFormatType>json</monFormatType>
</Monitor>
```

The response from this command is simply the empty result. However, by doing the GET again you can verify that your change was effective; the format type is now json.

GET /ws/Monitor/267

```
<result>
  <resultTotalRows>1</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>1</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <Monitor>
    <monId>267</monId>
    <cstId>27</cstId>
    <monTopic>DeviceCore,XbeeCore</monTopic>
    <monTransportType>tcp</monTransportType>
    <monFormatType>json</monFormatType>
    <monBatchSize>1</monBatchSize>
    <monCompression>none</monCompression>
    <monStatus>INACTIVE</monStatus>
    <monBatchDuration>0</monBatchDuration>
  </Monitor>
</result>
```

14.2.1.5 Remove a monitor using DELETE

To delete the monitor send the web services call of DELETE, referencing the number of the monitor. For example, DELETE /ws/Monitor/267.

To confirm that the monitor is now removed, send another GET. The result below shows that there are zero rows in the response, confirming that the monitor was removed.

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

Alternatively, a condition can be specified to delete all TCP monitors which are inactive. To do so send:

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

14.2.2 Example 2 - Creating an Advanced Monitor

The following example describes how to create a new monitor using one topic: FileData. In this example you will monitor only "update" operations pertaining to devices containing a common string. In the following example the [operation=U] text signifies all FileData update operations, while the. *40* text signifies all FileData devices whose Device ID contains the number "40" in the center of the ID number.

NOTE: The FileData topic is being used for demonstration purposes only. You can follow this example using any of the supported monitor topics.

14.2.2.1 Create new device monitor using POST

First, use a POST to set up the monitor:

POST /ws/Monitor

```
<Monitor>
  <monTopic>[operation=U]FileData/*.*40.*</monTopic>
  <monTransportType>tcp</monTransportType>
  <monFormatType>xml</monFormatType>
  <monBatchSize>1</monBatchSize>
  <monCompression>none</monCompression>
  <monBatchDuration>0</monBatchDuration>
</Monitor>
```

Which will return the following response:

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <location>Monitor/19037</location>
</result>
```

Notice that the response contains the monitor ID of 19037.

14.2.2.2 List configured monitors using GET

Now use a GET to see this monitor in place (the following is the response from the example GET). You can see that monitor 19037 is in place.

GET /ws/Monitor

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <resultTotalRows>1</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>1</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <Monitor>
```

```

<monId>19037</monId>
<csId>2</csId>
<monTopic>[operation=U]FileData/*.*40.*</monTopic>
<monTransportType>tcp</monTransportType>
<monFormatType>xml</monFormatType>
<monBatchSize>1</monBatchSize>
<monCompression>none</monCompression>
<monStatus>INACTIVE</monStatus>
<monBatchDuration>0</monBatchDuration>
</Monitor>
</result>

```

14.2.2.3 Modify an existing monitor using PUT

Next, perform an update using a monitor PUT. In this case, changing the format type from 'xml' to 'json'.

PUT /ws/Monitor

```

<Monitor>
  <monId>19037</monId>
  <monFormatType>json</monFormatType>
</Monitor>

```

The response from this command is simply the empty result. However, by doing the GET again you can verify that your change was effective; the format type is now json.

GET /ws/Monitor/19037

```

<result>
  <resultTotalRows>1</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>1</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <Monitor>
    <monId>19037</monId>
    <csId>2</csId>
    <monTopic>[operation=U]FileData/*.*40.*</monTopic>
    <monTransportType>tcp</monTransportType>
    <monFormatType>json</monFormatType>
    <monBatchSize>1</monBatchSize>
    <monCompression>none</monCompression>
    <monStatus>INACTIVE</monStatus>
    <monBatchDuration>0</monBatchDuration>
  </Monitor>
</result>

```

14.2.2.4 Remove a monitor using DELETE

To delete the monitor send the web services call of DELETE, referencing the number of the monitor. For example, DELETE /ws/Monitor/19037.

To confirm that the monitor is now removed, send another GET. The result below shows that there are zero rows in the response, confirming that the monitor was removed.

```

<result>
  <resultTotalRows>0</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>0</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
</result>

```

Alternatively, a condition can be specified to delete all TCP monitors which are inactive. To do so send:

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

14.3 Supported Monitor Topics

- Alarm
- AlarmStatus
- DataPoint
- DataStream
- DiaChannelDataFull
- DeviceCore
- FileData
- FileDataCore
- Job
- JobResult
- XbeeAttributeDataCore
- XbeeCore
- XbeeEventDataCore

NOTE: FileData and FileDataCore events will not be published when the file size is larger than 120K.

14.4 Topics

Topics are generally specified by the resource name followed optionally by the resource ID using the normal [REST](#) slash convention. For example:

- XbeeAttributeDataCore matches all XbeeAttributeData events reported under this user
- XbeeAttributeDataCore/23:11:87:a9:ca matches all XbeeAttributeData events reported by the node with xpExtAddr of 23:11:87:a9:ca
- XbeeAttributeDataCore/23:11:87:a9:ca/513 further narrows events down to endpoint 513

This convention is the same as that used to fetch resources through ws GET operations. As shown above, the separator character between topic components is the forward slash character '/'.

NOTE: Take care to [URL](#) encode the following special characters when specifying additional subtopic(ID) components:

- /
- %
- .
- *
- [
-]

When monitor topics are reported, these components will also be URL encoded. This allows for easier parsing of monitor topics. The general procedure is to split the published topic string on the '/' separator character and then URL decode the identified components.

You can monitor multiple topics by separating the topic strings with commas. For Example:

- DeviceCore,XbeeCore matches all operations against either DeviceCore or XbeeCore resources.

14.5 Batched Payloads

A monitor can be configured to batch messages together before delivering them to the client. As a result, each incoming monitor PublishEventMessage can have a payload containing multiple messages. For example:

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

or

```
{
  "Document":{
    "Msg":[
      {
        "DeviceCore":{
          "...": "..."
        },
        "group": "simulated",
        "operation": "DELETE",
        "timestamp": "2012-07-05T20:23:35.483Z",
        "topic": "8/DeviceCore/126667/0"
      },
      {
        "DeviceCore":{
          "...": "..."
        },
        "group": "simulated",
        "operation": "UPDATE",
        "timestamp": "2012-07-05T20:23:35.607Z",
        "topic": "8/DeviceCore/126668/0"
      },
      {
        "DeviceCore":{
          "...": "..."
        },
        "group": "simulated",
        "operation": "DELETE",
        "timestamp": "2012-07-05T20:23:35.772Z",
        "topic": "8/DeviceCore/126668/0"
      }
    ]
  }
}
```

14.6 Publish Order

Creates are published after the underlying create operation is persisted.

Updates are published after the underlying update operation is persisted.

Deletes are published before the underlying delete operation is persisted.

14.7 Batching and Compression

Customers can provide hints to Device Cloud on how they would like messages delivered to their application. This is done using the *monBatchSize*, *monBatchDuration*, and compression options. By aggregating and compressing messages before they are transmitted to the application, the messages can be delivered quicker and more efficiently. The *monBatchSize* parameter is used to place an upper bound on how many messages will be aggregated together before sending them to the client. The maximum *monBatchSize* that can be configured is 1000. The *monBatchDuration* parameter is used to place an upper bound on how many seconds messages will be aggregated for before sending them to the client. The maximum *monBatchDuration* is 3600 seconds, or 1 hour. The platform may dynamically alter these parameters to optimally deliver messages. Finally, the compression option indicates what method should be used to compress the messages.

14.8 Auto Replay of Missed Published Events

This option provides customers a way to recover events which were missed or lost due to a network disruption in the connection from Device Cloud to the customer's application. When this option is enabled, Device Cloud will record underlying published events which couldn't be successfully forwarded. Then, when the network connection is reestablished, Device Cloud will replay the missed events before forwarding any new publish events.

NOTE: For a new monitor, Device Cloud will wait to start recording events until the monitor has successfully connected for the first time.

To enable this feature, use the *monAutoReplayOnConnect* option in the monitor configuration:

```
<Monitor>
  <monTopic>DeviceCore,XbeeCore</monTopic>
  <monTransportType>http</monTransportType>
  <monTransportUrl>your website url</monTransportUrl>
  <monTransportToken>username:password</monTransportToken>
  <monFormatType>xml</monFormatType>
  <monBatchSize>1</monBatchSize>
  <monCompression>none</monCompression>
  <monBatchDuration>0</monBatchDuration>
  <monAutoReplayOnConnect>true</monAutoReplayOnConnect>
</Monitor>
```

The resent missed events will be indicated with the text **replay="true"** in the message envelope wrapper:

```
<Document>
  <Msg topic="3/DeviceCore/4/0" group="test2" operation="UPDATE" timestamp="2012-07-03T19:33:23.006Z" replay="true">
```

```

<DeviceCore>
  <id>
    <devId>4</devId>
    <devVersion>0</devVersion>
  </id>
  <!-- content shortened for the example -->
</DeviceCore>
</Msg>
</Document>

```

15. Scheduled Operations

NOTE: Before proceeding with this chapter you must have a thorough understanding of the information outlined in the "Managing Schedules" Chapter of the Device Cloud User's Guide. Please see the *Device Cloud User's Guide* (available within the Documentation tab) for more information.

15.1 Introduction

Building a chain of operations can be tedious, especially when you must build this chain every time you wish to run an operation. To streamline this process, Device Cloud gives users the ability to store chains of operations for future use. The Scheduled Operations feature makes it easy for you to create schedules in order to perform common management tasks on one or a group of devices. These tasks include updating a device(s) firmware, rebooting a device, uploading, retrieving, or deleting files, etc. Basically, any [SCI](#) request can be scheduled using this feature. When creating a schedule you also have the option of specifying whether it will be a one time or recurring schedule.

The scheduled operations feature consists of task templates, schedules and individual tasks:

- A **task** may consist of one or more commands chained together. Tasks are the actual commands which run according to a schedule. Tasks can be managed (created, deleted, scheduled, status queried, etc). When the task is executed, the task itself is assigned an ID which represents the status of the overall task. However, each individual command specified within the task's task template is executed as a job within the system. When a series of jobs is scheduled for the same device, one job does not start until the previous job in the series has been completed.
- A **task template** is the framework of a schedule; it specifies each of the management commands you want to run. This template can contain a single command or a series of commands to be run sequentially. Saved task templates are stored as [XML](#) files. These files can be found within the my_tasks folder of the Data Services page within Device Cloud.

NOTE: See the *Device Cloud User's Guide* for information regarding Data Streams.

- A **schedule** specifies when the chosen commands (outlined within the task template) will be executed and which device(s) will be targeted. When creating a schedule you will need to specify when the chosen commands will be executed by selecting a schedule type (immediate, one-time, or recurring). You will also need to select which device(s) will be targeted. You have the option of selecting one or more devices to target, and devices can be targeted individually, by Tag name, by Group name, or any combination of these three methods.

This feature also supports the following: Offline Operations and Wait for Reconnect

NOTE: See the [Available Operations](#) section for more information on these features.

15.2 Task Template Overview

A task template is the framework of a schedule; it specifies each of the management commands that will be included in a schedule. This template can contain a single command or a series of commands to be run sequentially, and each command contains its their specific command payload ([SCI](#)) and events. Each command within a task template is executed sequentially based on the order that it is listed.

The following is an example of a task template which contains two commands (this example's elements are described in more detail within the [Elements of a Task Template](#) section).

```

<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>
    <name>Remove Python File</name>
    <event>
      <on_error>
        <continue />
      </on_error>
    </event>
    <sci>
      <send_message allowOffline="true">
        <rci_request version="1.1">
          <do_command target="file_system">
            <rm name="/WEB/python/somefile.py" />
          </do_command>
        </rci_request>
      </send_message>
    </sci>
  </command>
</task>

```

15.2.1 Elements of a Task Template

15.2.1.1 Description

The description element allows you to give a descriptive name to your task template.

```
<description>My first task.</description>
```

15.2.1.2 Command

The command element contains the configuration information for each of the commands in the chain. If your task template contains multiple commands, they will be listed sequentially. This list determines the order in which the individual commands will be executed.

```

<command>
  <name>Reboot</name>

```

```

<event>
  <on_end>
    <sleep value="15" />
  </on_end>
  <on_error>
    <retry count="5" />
  </on_error>
</event>
<sci>
  <reboot />
</sci>
</command>

```

15.2.1.3 Name

The name element allows you to provide a name for each of the commands in the chain. This element is not required. If you choose not to utilize this element, simply do not include this line in your task template.

```
<name>Reboot</name>
```

15.2.1.4 Events

The Scheduled Operations feature currently supports two types of events: `on_end` and `on_error`.

On End

The On End element specifies a sleep value (in seconds) before moving onto the next command in the chain. The Sleep option will force Device Cloud to delay the execution of the next command in the list for the specified number of seconds, immediately following the completion of the command. For example, the example below would cause Device Cloud to delay the execution of the next command in the list for 15 seconds. This happens on a per-command basis. Therefore, if you have this setting enabled for multiple commands within your schedule, the server will sleep for the specified number of seconds following the completion of each of the commands that has this setting enabled. If you choose not to utilize the Sleep option, simply do not include this line in your task template.

```
<on_end> <sleep value="15"/> </on_end>
```

On Error

The On Error element specifies what action should be taken if an error occurs while the chain of commands is being executed. Depending on which option is selected, the chain of commands will end immediately, continue with the next command in the chain, or retry for a specified number of times.

```

<on_error>
  <retry count="5" />
  or
  <end_task />
  or
  <continue />
</on_error>

```

The **End Task** option will end the task immediately in the event of an error, and no other commands within your schedule will be executed. If an error occurs when attempting to execute a command and the End Task option is selected for that command, your schedule will fail.

```

<on_error>
  <end_task />
</on_error>

```

The **Continue** option will proceed with the next command in the list in the event that the current command fails.

```

<on_error>
  <continue />
</on_error>

```

The **Retry** option will allow you to specify a number of retry attempts. For example, the example below would allow for 5 retry attempts. If a command is successfully executed before the number of retry attempts is exceeded, the schedule will continue and the next command in the list will be executed. If a command exceeds the number of retry attempts without success, the task will fail and no other commands in your schedule will be executed.

```

<on_error>
  <retry count="5" />
</on_error>

```

15.3 Schedule API

The Schedule API allows you to schedule the execution of a task template. A task template can be either a file in storage or embedded in the payload.

When creating or modifying a schedule, three attributes need to be specified:

on - When to run the task. Available options are "IMMEDIATE", ISO 8601 date or ISO 8601 duration.

targets - What devices, tags or group is the task targeted to.

task - What task to run. This can either be a file in storage or can be embedded in the body itself.

```

<schedule on="2012-01-01T16:00:00-06:00Z">
  <targets>
    <device id="00000000-00000000-00000000-00000000" />
  </targets>
  <task>
    <description>My first task.</description>
    <command>
      <name>Reboot</name>
      <event>
        <on_end>
          <sleep value="15" />
        </on_end>
        <on_error>
          <retry count="5" /> | <end_task /> | <continue />
        </on_error>
      </event>
      <sci>
        <reboot />
      </sci>
    </command>

    <command>
      <name>Remove Python File</name>
      <event>

```

```

        <on_error>
        <continue />
        </on_error>
    </event>
    <sci>
        <send_message allowOffline="true">
            <rci_request version="1.1">
                <do_command target="file_system">
                    <rm name="/WEB/python/somefile.py" />
                </do_command>
            </rci_request>
        </send_message>
    </sci>
</command>
</task>
</schedule>

```

15.3.1 Elements of a Schedule

15.3.1.1 Scheduling Attributes

Scheduling attributes determine when the schedule will be executed. Depending on what values are configured, the schedule will be executed immediately or on a specified date.

Date and Duration attributes must be expressed using ISO 8601 format. ISO 8601 is the International Standard for the representation of dates and times. In order to understand the format of the On, Start, and End scheduling attributes (listed below), you must first understand how to properly express Date and Duration attributes in ISO 8601 format. Please refer to the [ISO 8601 Date & Duration Reference](#) Appendix for more information on expressing Date and Duration attributes in ISO 8601 format.

On

The on attribute states when the task will run. Available options are:

- "IMMEDIATE" - will immediately execute the task.
- ISO 8601 date - will execute the schedule on the given start date.
- ISO 8601 (repeat) duration - will repeat the execution of the schedule for the given duration.

Start

The start attribute pertains to recurring schedules. This value states when a recurring schedule will start.

```
<schedule on="PT1H" start="2012-07-01T16:00:00-06:00Z" > <!-- This schedule will run, every hour, after the given start ISO 8601 Date -->
```

End

The end attribute pertains to recurring schedules. This value states when a recurring schedule will end.

```
<schedule on="PT1H" end="2012-07-01T16:00:00-06:00Z" > <!-- This schedule will run, every hour starting immediately, up until the given end ISO 8601 Date -->
```

Start and End (repeat)

The start attribute pertains to recurring schedules. This value states when a recurring schedule will start and end.

```
<schedule on="PT4H" start="2012-05-01T16:00:00-06:00Z" end="2012-09-01T16:00:00-06:00Z" > <!-- This schedule will repeat every 4 hours, between the start ISO 8601 Date and end ISO 8601 Date -->
```

15.3.1.2 Targets

The target element specifies which devices, tags, or group the task is targeted towards. You have the option of selecting one or more devices to target, and devices can be targeted individually, by Tag name, by Group name, or any combination of these three methods.

```

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

```

15.3.1.3 Task Template

A task template is the framework of a schedule; it specifies each of the management commands that will be included in a schedule. This template can contain a single command or a series of commands to be run sequentially, and each command contains its specific command payload (SCI) and events. Each command within a task template is executed sequentially based on the order that it is listed.

NOTE: For an example of a task template see the [Task Template Overview](#) section. For detailed information on the elements of a task template see the [Elements of a Task Template](#) section.

An **embedded** task template allows you to embed an entire task template into your schedule.

A **referenced** task template allows you to reference the path of the task template you'd like to include in your schedule.

```

<operation>
  <targets>
    <device id="00000000-00000000-00000000-00000000" />
  </targets>
  <task path="/db/my_new_task.xml" /> <!--or embed the entire task template -->
</operation>

```

POST

POST schedules a task template using the elements described in the previous section.

Example:

POST /ws/Schedule

```

<schedule on="DATE | IMMEDIATE | DURATION" start="DATE" end="DATE">
  <operation>
    <targets>
      <device id="00000000-00000000-00000000-00000001" />
      <device id="00000000-00000000-00000000-00000002" />
    </targets>
    <task path="/db/my_new_task.xml" /> <!--or embed the entire task template -->
  </operation>
</schedule>

```

where the on attribute can be:

1. Date (in ISO8601 date format)
For example, 2012-01-01T16:00:00-06:00Z //January first of 2012 at 4:00 PM GMT-6:00
2. IMMEDIATE (just a string)
For example, IMMEDIATE //Runs right away

3. DURATION (in ISO8601 Duration format)
For example, PT4H //Runs every 4 hours

GET

GET is used to retrieve a list of schedules, or a specific schedule by its ID.

Elements include:

- schId - ID of the schedule
- schDescription - description name given to the schedule
- schExpression - the expression used to define when the schedule will run (e.g. 'IMMEDIATE')
- schTargets - the list of targets the schedule is targeted towards
- schStartTime - actual schedule execution time
- schStopTime - schedule end time
- schStatus - current status of the schedule (0: new, 1: in_progress, 3: complete, 4 canceled)
- schPreviousRunTime - actual time when the schedule was last executed
- task - the task template associated with the schedule

Example #1:

To get a list of all schedules:

GET ws/Schedule

Returns a list of all configured schedules.

Example #2:

To get the details of a specific schedule ID:

GET ws/Schedule/{schId}

Returns schedule details for the specified schedule ID.

PUT

PUT allows you to modify a schedule's attributes.

Example:

For example, if you have a schedule that is set to run every 12 hours but you'd like to modify it to run every 4 hours:

PUT ws/Schedule/{schId}

```
<schedule on="PT4H">
  <targets>
    <device id="00000000-00000000-00000000-00000000" />
  </targets>
</schedule>
```

This will change the schedule element (along with its potential attributes) and modify the schedule to run every 4 hours.

DELETE

DELETE is used to delete a specified schedule.

Example:

To delete a schedule:

DELETE ws/Schedule/{schId}

Will delete the schedule matching the specified schedule ID.

15.4 Task API

The Task API allows you to execute a specific task. A task is a chain of commands stored in Device Cloud's file system (as an XML file). Each command element 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. These tasks can be managed through the Task API. These tasks, which execute commands, create [SCI](#) jobs within the system on a per-command basis. It is possible to see the status of each command by querying the [SCI](#) web service interface for a given job.

GET

GET is used to retrieve a list of tasks, or a specific task by its ID.

Elements include:

- taskId - the ID of the task.
- schId - the ID of the schedule that created this task.
- taskScheduledTime - the time when this task was scheduled to run.
- taskStartTime - actual task execution time.
- taskEndTime - task end time.
- taskTargets - the list of targets the task is targeted towards.
- taskSuccess - the number of devices that have completed the task successfully.
- taskFailures - the number of devices that have completed the task with an error.
- taskStatus - current status of the task (0: new, 1: in_progress, 2: complete, 3: canceled, 4: failed).
- taskRequestPayload - the request payload of the task.
- taskTargetCount - total number of devices the task is targeted to.

Example #1:

To get a list of all tasks:

GET ws/Task

Returns a list of tasks.

Example #2:

To get a list of all the tasks associated with a specific schedule ID:

GET ws/Task?condition=schId={schId}

Returns details for each of the tasks associated with the specified schedule ID.

Example #3:

To get the details of a specific task ID:

GET ws/Task/{taskId}

Returns details for the specified task ID.

Example #4:

To get a list of all the jobs associated with a specific schedule ID:

GET ws/sci?condition=schId={schId}

Returns a list of all the jobs created by the specified schedule ID.

Example #5:

To get detailed payload information for a job:

GET ws/sci/{id}

Returns specific payload and other information related to the specified job.

PUT

PUT allows you to upload a task definition.

Example:

PUT ws/FileData/~ /my_tasks/my_task.xml?type=file

DELETE

DELETE is used to delete a task definition.

Example:

To delete a task:

DELETE ws/FileData/~ /my_tasks/my_task.xml

15.5 Successful Device Reboot Example

The following example describes how to immediately schedule your device to reboot. In this example you will use the Schedule and Task APIs to post the schedule to your device, and then go through the necessary steps of determining whether or not your device successfully rebooted.

1.Set up the schedule using POST

First, use a POST to set up the schedule:

Post to ws/Schedule

```
<schedule on="IMMEDIATE">
  <targets>
    <device id="00000000-00000000-0004F3FF-FF03A80C" />
  </targets>
  <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>
  </task>
</schedule>
```

Which will return the following response:

```
<Schedule>
  <schId>1523</schId>
</Schedule>
```

Notice that the response contains the schedule ID of 1523.

2.List the details of a schedule using GET

Next use a GET to return the details of this schedule. This can be done using the Schedule API or the Task API:

-To return the schedule details for the specified schedule ID (using the Schedule API), perform a GET on /ws/Schedule/1523:

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <resultTotalRows>1</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>1</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <Schedule>
    <schId>1523</schId>
    <cstId>2</cstId>
    <usrId>8</usrId>
    <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>
    </task>
    <schDescription>My first task.</schDescription>
    <schExpression>IMMEDIATE</schExpression>
    <targets>
      <device id="00000000-00000000-0004F3FF-FF03A80C" />
    </targets>
    <schStartTime>2012-03-28T18:28:42.553Z</schStartTime>
    <schStopTime>2012-03-28T18:28:42.553Z</schStopTime>
    <schStatus>3</schStatus>
    <schPreviousRunTime>2012-03-28T18:28:42.553Z</schPreviousRunTime>
```

```
</Schedule>
</result>
```

-To return the details for each of the tasks associated with the specified schedule ID (using the Task [API](#)), perform a GET on /ws/Task?condition=schId=1523:

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <resultTotalRows>1</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>1</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <Task>
    <tskId>35225</tskId>
    <schId>1523</schId>
    <csId>2</csId>
    <usrId>8</usrId>
    <tskScheduledTime>2012-03-28T18:28:42.553Z</tskScheduledTime>
    <tskStartTime>2012-03-28T18:28:42.867Z</tskStartTime>
    <tskEndTime>2012-03-28T18:28:58.107Z</tskEndTime>
    <tskTargets>00000000-00000000-0004F3FF-FF03A80C</tskTargets>
    <tskSuccess>1</tskSuccess>    <!-- 1 successful task has occurred -->
    <tskFailures>0</tskFailures>
    <tskStatus>2</tskStatus>
    <tskRequestPayload>
      <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>
    </tskRequestPayload>
    <tskTargetCount>1</tskTargetCount>
    <tskDescription>My first task.</tskDescription>
  </Task>
</result>
```

If you observe the tskSuccess line (indicated in the example above) you will see that 1 successful task has occurred, which would imply that this task was successfully executed. However, to be certain that the task was successful you can further examine the schedule's details.

Looking at this task we see that its task Id is 35225 generated via schedule 1523. We will use this ID to verify if the task was successfully executed or not.

3.List all the jobs associated with a task using GET

Performing a Get on /ws/sci?condition=tskId=35225 will return back all of the jobs created by this task:

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <resultTotalRows>1</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>1</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <Job>
    <jobId>8660668</jobId>
    <csId>2</csId>
    <usrId>8</usrId>
    <tskId>35225</tskId>
    <jobType>12</jobType>
    <jobSyncMode>1</jobSyncMode>
    <jobReplyMode>0</jobReplyMode>
    <jobTargets>00000000-00000000-0004F3FF-FF03A80C</jobTargets>
    <jobRequestPayload />
    <jobDescription>reboot</jobDescription>
    <jobStatus>2</jobStatus>
    <jobTargetCount>1</jobTargetCount>
    <jobProgressSuccess>1</jobProgressSuccess>
    <jobProgressFailures>0</jobProgressFailures>
    <jobSubmitTime>2012-03-28T18:28:42.927Z</jobSubmitTime>
    <jobCompletionTime>2012-03-28T18:28:43.077Z</jobCompletionTime>
    <jobOfflineMode>1</jobOfflineMode>
    <cmdId>193903</cmdId>
  </Job>
</result>
```

Notice that the response contains the job ID of 8660668. We can use this job ID to verify whether or not the task was completed successfully.

4.View a job's status using GET

To view the status of job 8660668, perform a Get on /ws/sci/8660668:

```
<sci_reply version="1.0">
  <status>complete</status>
  <reboot>
    <device id="00000000-00000000-0004F3FF-FF03A80C">
      <reboot />
    </device>
  </reboot>
</sci_reply>
```

No failures are listed for the job, indicating that the device was successfully rebooted.

15.6 Unsuccessful Device Reboot Example

The following example describes how to immediately schedule your device to reboot. In this example you will use the Schedule and Task APIs to post the schedule to your device, and then go through the necessary steps of determining whether or not your device successfully rebooted.

1.Set up the schedule using POST

First, use a POST to set up the schedule:

Post to ws/Schedule

```
<schedule on="IMMEDIATE">
  <targets>
    <device id="00000000-00000000-9100A0FF-FF00001D" />
  </targets>
  <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>
  </task>
</schedule>
```

Which will return the following response:

```
<Schedule>
  <schId>326</schId>
</Schedule>
```

Notice that the response contains the schedule ID of 326.

2.List the details of a schedule using GET

Next use a GET to return the details of this schedule. This can be done using the [Schedule API](#) or the [Task API](#):

-To return the schedule details for the specified schedule ID (using the [Schedule API](#)), perform a GET on [/ws/Schedule/326](#):

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <resultTotalRows>1</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>1</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <Schedule>
    <schId>326</schId>
    <csiId>2</csiId>
    <usrId>9</usrId>
    <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>
    </task>
    <schDescription>My first task.</schDescription>
    <schExpression>IMMEDIATE</schExpression>
    <targets>
      <device id="00000000-00000000-9100A0FF-FF00001D" />
    </targets>
    <schStartTime>2012-03-26T18:27:56.300Z</schStartTime>
    <schStopTime>2012-03-26T18:27:56.300Z</schStopTime>
    <schStatus>3</schStatus>
    <schPreviousRunTime>2012-03-26T18:27:56.300Z</schPreviousRunTime>
  </Schedule>
</result>
```

-To return the details for each of the tasks associated with the specified schedule ID (using the [Task API](#)), perform a GET on [/ws/Task?condition=schId=326](#):

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <resultTotalRows>1</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>1</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <Task>
    <tskId>19114</tskId>
    <schId>326</schId>
    <csiId>2</csiId>
    <usrId>9</usrId>
    <tskScheduledTime>2012-03-26T18:27:56.300Z</tskScheduledTime>
    <tskStartTime>2012-03-26T18:27:57.837Z</tskStartTime>
    <tskEndTime>2012-03-26T18:28:40.237Z</tskEndTime>
    <tskTargets>00000000-00000000-9100A0FF-FF00001D</tskTargets>
    <tskSuccess>0</tskSuccess>
    <tskFailures>1</tskFailures> <!-- 1 failure has occurred -->
    <tskStatus>2</tskStatus>
    <tskRequestPayload>
      <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>
</taskRequestPayload>
<taskTargetCount>1</taskTargetCount>
<taskDescription>My first task.</taskDescription>
</Task>
</result>

```

If you observe the taskFailures lines (indicated in the example above) you will see that 1 failure occurred, which would imply that this task failed. However, to be certain that the task failed and to determine the cause of this failure you can further examine the schedule's details.

Looking at this task we see that its task Id is 19114 generated via schedule 326. We will use this ID to verify whether or not the task failed, and determine the cause of this failure.

3.List all the jobs associated with a task using GET

Performing a Get on /ws/sci?condition=tskId=19114 will return back all of the jobs created by this task:

```

<?xml version="1.0" encoding="UTF-8"?>
<result>
  <resultTotalRows>6</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>6</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <Job>
    <jobId>96711</jobId>
    <csiId>2</csiId>
    <usrId>9</usrId>
    <tskId>19114</tskId>
    <jobType>12</jobType>
    <jobSyncMode>1</jobSyncMode>
    <jobReplyMode>0</jobReplyMode>
    <jobTargets>00000000-00000000-9100A0FF-FF00001D</jobTargets>
    <jobRequestPayload />
    <jobDescription>reboot</jobDescription>
    <jobStatus>2</jobStatus>
    <jobTargetCount>1</jobTargetCount>
    <jobProgressSuccess>0</jobProgressSuccess>
    <jobProgressFailures>1</jobProgressFailures>
    <jobSubmitTime>2012-03-26T18:27:58.207Z</jobSubmitTime>
    <jobCompletionTime>2012-03-26T18:27:59.630Z</jobCompletionTime>
    <jobOfflineMode>1</jobOfflineMode>
    <cmdId>19559</cmdId>
    <jobRetryId>96712</jobRetryId>
    <jobNextId>96712</jobNextId>
  </Job>
  <Job>
    <jobId>96712</jobId>
    <csiId>2</csiId>
    <usrId>9</usrId>
    <tskId>19114</tskId>
    <jobType>12</jobType>
    <jobSyncMode>1</jobSyncMode>
    <jobReplyMode>0</jobReplyMode>
    <jobTargets>00000000-00000000-9100A0FF-FF00001D</jobTargets>
    <jobRequestPayload />
    <jobDescription>reboot</jobDescription>
    <jobStatus>2</jobStatus>
    <jobTargetCount>1</jobTargetCount>
    <jobProgressSuccess>0</jobProgressSuccess>
    <jobProgressFailures>1</jobProgressFailures>
    <jobSubmitTime>2012-03-26T18:28:04.863Z</jobSubmitTime>
    <jobCompletionTime>2012-03-26T18:28:04.937Z</jobCompletionTime>
    <jobOfflineMode>1</jobOfflineMode>
    <cmdId>19559</cmdId>
    <jobRetryId>96713</jobRetryId>
    <jobNextId>96713</jobNextId>
  </Job>
  <Job>
    <jobId>96713</jobId>
    <csiId>2</csiId>
    <usrId>9</usrId>
    <tskId>19114</tskId>
    <jobType>12</jobType>
    <jobSyncMode>1</jobSyncMode>
    <jobReplyMode>0</jobReplyMode>
    <jobTargets>00000000-00000000-9100A0FF-FF00001D</jobTargets>
    <jobRequestPayload />
    <jobDescription>reboot</jobDescription>
    <jobStatus>2</jobStatus>
    <jobTargetCount>1</jobTargetCount>
    <jobProgressSuccess>0</jobProgressSuccess>
    <jobProgressFailures>1</jobProgressFailures>
    <jobSubmitTime>2012-03-26T18:28:09.960Z</jobSubmitTime>
    <jobCompletionTime>2012-03-26T18:28:09.993Z</jobCompletionTime>
    <jobOfflineMode>1</jobOfflineMode>
    <cmdId>19559</cmdId>
    <jobRetryId>96714</jobRetryId>
    <jobNextId>96714</jobNextId>
  </Job>
  <Job>
    <jobId>96714</jobId>
    <csiId>2</csiId>
    <usrId>9</usrId>
    <tskId>19114</tskId>
    <jobType>12</jobType>
    <jobSyncMode>1</jobSyncMode>
    <jobReplyMode>0</jobReplyMode>
    <jobTargets>00000000-00000000-9100A0FF-FF00001D</jobTargets>
    <jobRequestPayload />
    <jobDescription>reboot</jobDescription>
    <jobStatus>2</jobStatus>
  </Job>

```

```

<jobTargetCount>1</jobTargetCount>
<jobProgressSuccess>0</jobProgressSuccess>
<jobProgressFailures>1</jobProgressFailures>
<jobSubmitTime>2012-03-26T18:28:15.027Z</jobSubmitTime>
<jobCompletionTime>2012-03-26T18:28:15.060Z</jobCompletionTime>
<jobOfflineMode>1</jobOfflineMode>
<cmdId>19559</cmdId>
<jobRetryId>96715</jobRetryId>
<jobNextId>96715</jobNextId>
</Job>
<Job>
  <jobId>96715</jobId>
  <csId>2</csId>
  <usrId>9</usrId>
  <tskId>19114</tskId>
  <jobType>12</jobType>
  <jobSyncMode>1</jobSyncMode>
  <jobReplyMode>0</jobReplyMode>
  <jobTargets>00000000-00000000-9100A0FF-FF00001D</jobTargets>
  <jobRequestPayload />
  <jobDescription>reboot</jobDescription>
  <jobStatus>2</jobStatus>
  <jobTargetCount>1</jobTargetCount>
  <jobProgressSuccess>0</jobProgressSuccess>
  <jobProgressFailures>1</jobProgressFailures>
  <jobSubmitTime>2012-03-26T18:28:20.120Z</jobSubmitTime>
  <jobCompletionTime>2012-03-26T18:28:20.147Z</jobCompletionTime>
  <jobOfflineMode>1</jobOfflineMode>
  <cmdId>19559</cmdId>
  <jobRetryId>96716</jobRetryId>
  <jobNextId>96716</jobNextId>
</Job>
<Job>
  <jobId>96716</jobId>
  <csId>2</csId>
  <usrId>9</usrId>
  <tskId>19114</tskId>
  <jobType>12</jobType>
  <jobSyncMode>1</jobSyncMode>
  <jobReplyMode>0</jobReplyMode>
  <jobTargets>00000000-00000000-9100A0FF-FF00001D</jobTargets>
  <jobRequestPayload />
  <jobDescription>reboot</jobDescription>
  <jobStatus>2</jobStatus>
  <jobTargetCount>1</jobTargetCount>
  <jobProgressSuccess>0</jobProgressSuccess>
  <jobProgressFailures>1</jobProgressFailures>
  <jobSubmitTime>2012-03-26T18:28:25.187Z</jobSubmitTime>
  <jobCompletionTime>2012-03-26T18:28:25.220Z</jobCompletionTime>
  <jobOfflineMode>1</jobOfflineMode>
  <cmdId>19559</cmdId>
</Job>
</result>

```

Since our initial schedule's task template had a retry count of 5 defined, and the entire task failed, the result above shows that the task was retried 5 times before fully failing.

Notice that the response contains several job IDs. We can use any of these job IDs, for example 96716, to determine why the overall task failed.

4. View a job's status using GET

To determine why this specific job failed, perform a Get on /ws/sci/96716:

```

<sci_reply version="1.0">
  <status>complete</status>
  <reboot>
    <device id="00000000-00000000-9100A0FF-FF00001D">
      <error id="2001">
        <desc>Device Not Connected</desc>
      </error>
    </device>
  </reboot>
</sci_reply>

```

Notice that the response contains an error stating that the device was not connected when the task was attempting to be executed. Since the device was not connected it caused this particular job (jobId=96716) which was created via a task (tskId=19114) contained within a scheduled (schId=326) set to run immediately to fail.

16. Alarms

NOTE: Before proceeding with this chapter you need to create one or more alarms within your Device Cloud account. To learn more about the alarms feature; including how to create a new alarm, see the "Managing Alarms" Chapter of the *Device Cloud User's Guide* (available within the Documentation tab).

16.1 Introduction

The Alarm Web Service is used to retrieve information about the alarms within your Device Cloud account. You have the option of retrieving information about all of the alarms within your account, or a single alarm. The alarms feature allows you to configure, manage, and react to alarm conditions in Device Cloud. Alarms can be defined to monitor various activities within Device Cloud such as:

- When a device or XBee Node disconnects and fails to reconnect within a specified amount of time
- XBee Nodes with an excessive number of deactivations or devices with an excessive number of disconnects
- Thresholds in device related data points or data points for DIA channel and Smart Energy attribute data
- Usage thresholds pertaining to subscriptions (usage for a carrier integrated service provider or a Device Cloud Service you've subscribed to)
- Missing data points or when devices have not reported DIA channel or Smart Energy data within a specified amount time

Once you have created an alarm targeting a single device, an XBee Node associated with a device, a group of devices, or a resource you can gather information about that alarm using the Alarm Web Service.

16.2 Alarm

The Alarm Web Service is used to retrieve information about the alarms within your Device Cloud account.

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

HTTP operations supported:

- GET: Get a list of your existing alarms

Elements include:

- **almId** - alarm ID of the alarm
- **cstId** - the automatically generated ID assigned to a customer
- **almId** - the ID of the alarm template used to create the alarm
- **grpId** - the automatically generated ID assigned to a customer's group
- **almName** - name of the alarm
- **almDescription** - basic alarm description
- **almEnabled** - a boolean indicating whether or not the alarm is enabled
- **almPriority** - the user defined alarm priority (high, medium, or low)
- **almScopeConfig** - Specifies a resource (Device, XbeeNode, Group, DataStream) the alarm is applied to; a scope name and a value are required. The following is a list of all available alarm scopes. Available scoping for the alarm depends upon the alarm type. Each alarm supports a subset of the following list:
 - **Group** - Applies an alarm to a group of devices. The value for this scope is a full group path.
 - **Device** - Applies an alarm to a single device. The value is the device's Device ID. For example, 00000000-00000000-00000000-00000000.
 - **XbeeNode** - Applies an alarm to an XbeeNode. The value is the XbeeNode extended address. For example, 00:13:A2:00:00:00:00:00.
 - **Resource** - Applies an alarm to a data stream path or a pattern. Additionally, resource scoping can include a * to match any element in the data stream path. For example, dia/channel/lth/temp matches all lth temperature reading for all devices.
 - **Global** - Applies an alarm at the customer level to monitor all instances of an entity, for example an alarm can be created to monitor the total number of web services calls for your account. This scoping option is currently only available for Subscription Usage alarms.
- **almRuleConfig** - Specifies the rule configuration for an alarm. This element contains two configurations: FireRule and ResetRule. FireRule specifies the condition to fire an alarm while ResetRule specifies the condition to reset a fired alarm. FireRule and ResetRule consist of a list of Variable elements. Each Variable element requires a name and value attribute. Available variable names will depend upon the type of alarm.

Xml structure for almScopeConfig:

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

NOTE: Only one <ScopingOptions> tag under <almScopeConfig> and one <Scope> tag under <ScopingOptions> can be supplied at a time.

Xml structure for almRuleConfig:

```
<almRuleConfig>
  <Rules>
    <FireRule name="fireRule1">
      <Variable name="vOperator" value="&gt;"/>
      <Variable name="vDataPointValue" value="1"/>
      <Variable name="vTimeUnit" value="seconds"/>
      <Variable name="vTimeout" value="10"/>
      <Variable name="vType" value="double"/>
    </FireRule>
    <ResetRule name="resetRule1">
      <Variable name="vrType" value="double"/>
      <Variable name="vrDataPointValue" value="1"/>
      <Variable name="vrOperator" value="&lt;="/>
      <Variable name="vrTimeUnit" value="seconds"/>
      <Variable name="vrTimeout" value="10"/>
    </ResetRule>
  </Rules>
</almRuleConfig>
```

Examples:

Example #1:

To get a list of all alarms:

GET ws/Alarm

This will return a list of all of the alarms configured within your Device Cloud account.

The example below contains three alarms (as indicated by the three almId entries below). Each of these entries contains different specifications regarding that alarm's specific purpose.

```
<result>
<resultTotalRows>3</resultTotalRows>
<requestedStartRow>0</requestedStartRow>
<resultSize>3</resultSize>
<requestedSize>1000</requestedSize>
<remainingSize>0</remainingSize>
<Alarm>
  <almId>142</almId>    <!-- alarm #1 -->
  <cstId>2</cstId>
  <almId>4</almId>
  <grpId>2</grpId>
  <almName>Device Excessive Connections</almName>
  <almDescription>Detects devices with an excessive number of connection
attempts.</almDescription>
  <almEnabled>true</almEnabled>
  <almPriority>0</almPriority>
  <almScopeConfig>
    <ScopingOptions>
      <Scope name="Group" value="/CUS0000001_Digi_International/" />
    </ScopingOptions>
  </almScopeConfig>
  <almRuleConfig>
    <Rules>
      <FireRule name="fireRule1">
        <Variable name="vConnectWindow" value="5" />
        <Variable name="vConnectCount" value="5" />
      </FireRule>
      <ResetRule name="resetRule1">
        <Variable name="vReconnectWindow" value="5" />
      </ResetRule>
    </Rules>
  </almRuleConfig>
</Alarm>
<Alarm>
  <almId>151</almId>    <!-- alarm #2 -->
  <cstId>2</cstId>
  <almId>2</almId>
  <grpId>2</grpId>
  <almName>Device Offline</almName>
  <almDescription>Detects when a device disconnects from Device Cloud and fails to reconnect
```

```

within the specified time</almDescription>
  <almEnabled>true</almEnabled>
  <almPriority>1</almPriority>
  <almScopeConfig>
    <ScopingOptions>
      <Scope name="Group" value="/CUS0000001_Digi_International/" />
    </ScopingOptions>
  </almScopeConfig>
  <almRuleConfig>
    <Rules>
      <FireRule name="fireRule1">
        <Variable name="vDeviceReconnectWindowDuration" value="10" />
      </FireRule>
      <ResetRule name="resetRule1" />
    </Rules>
  </almRuleConfig>
</Alarm>
<Alarm>
  <almId>152</almId>      <!-- alarm #3 -->
  <csId>2</csId>
  <almId>1</almId>
  <grpId>2</grpId>
  <almName>SystemAlarm</almName>
  <almDescription>Detects when system alarm conditions occur</almDescription>
  <almEnabled>true</almEnabled>
  <almPriority>0</almPriority>
  <almScopeConfig>
    <almScopeConfig />
  </almScopeConfig>
  <almRuleConfig>
    <almRuleConfig />
  </almRuleConfig>
</Alarm>
</result>

```

Example #2:

To get the details of a specific alarm ID:

GET ws/Alarm/{almId}

16.3 AlarmStatus

The AlarmStatus Web Service is used to retrieve the current status of an alarm.

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

HTTP operations supported:

- GET: Get a list of current alarm statuses

Elements include:

- id
 - almId - alarm ID of the alarm
 - almSourceEntityId - the specific device or source entity of this alarm status
- almsStatus - current status of the alarm (0: reset, 1: triggered, 2: acknowledged)
- almsTopic - the alarm topic
- csId - the automatically generated ID assigned to a customer
- almsUpdateTime - the time the status was last updated
- almsPayload - the payload associated with the status change of this alarm in XML format. This can be any event in the system that caused the status of the alarm to change. As a result this value is highly varied, but almost always is some other object already represented and documented in Web Services.

Examples:

Example #1:

To get a list of all current alarm statuses:

GET ws/AlarmStatus

This will return a list of all of the alarms statuses for all of your configured alarms.

The example below contains four alarm statuses; two for alarm 142, one for alarm 151, and one for alarm 152.

```

<?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>
    <csId>2</csId>
    <almsUpdateTime>2012-06-27T21:02:09.567Z</almsUpdateTime>
    <almsPayload>
      <Payload>
        <Message>Monitor disconnected: node left the cluster</Message>
        <Monitor>
          <monId>46911</monId>
          <csId>2</csId>
          <monLastConnect>2012-06-27T17:08:27.457Z</monLastConnect>
          <monTopic>AlarmTemplate,Alarm,AlarmStatus,DeviceCore,XbeeCore</monTopic>
          <monTransportType>alarm</monTransportType>
          <monFormatType>xml</monFormatType>
          <monBatchSize>1</monBatchSize>
          <monCompression>none</monCompression>
          <monStatus>1</monStatus>
          <monBatchDuration>0</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>1</monBatchSize>
        <monCompression>none</monCompression>
        <monStatus>0</monStatus>
        <monBatchDuration>0</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>
        <dpCapabilities>2618</dpCapabilities>
        <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</monTopic>
        <monTransportType>alarm</monTransportType>
        <monFormatType>xml</monFormatType>
        <monBatchSize>1</monBatchSize>
        <monCompression>none</monCompression>
        <monStatus>1</monStatus>
        <monBatchDuration>0</monBatchDuration>
      </Monitor>
    </Payload>
  </almsPayload>
</AlarmStatus>
</result>

```

Example #2:

To get the status details of a specific alarm ID:

GET ws/AlarmStatus/{almId}

16.4 AlarmStatusHistory

The AlarmStatusHistory Web Service is used to retrieve a record of alarm statuses over time.

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

HTTP operations supported:

- GET: Get a list of alarm statuses over time

Elements include:

- id
 - almId - alarm ID of the alarm
 - almSourceEntityId - the specific device or source entity of this alarm status
- almsStatus - current status of the alarm
- almsTopic - the alarm topic
- cstId - the automatically generated ID assigned to a customer
- almsUpdateTime - the time the status was last updated
- almsPayload - the payload associated with the status change of this alarm in XML format. This can be any event in the system that caused the status of the alarm to change. As a result this value is highly varied, but almost always is some other object already represented and documented in web services.

Query parameters:

- size - number of resources to return in a GET request (default: 1000, max: 1000)
- pageCursor - page cursor returned from a previous request that can be used to retrieve the next page of data
- startTime - the start time (inclusive)
- endTime - the end time (exclusive)
- order - ascending or descending (actually checks if parameter value starts with 'asc' or 'desc')

Examples:

Example #1:

To get a list of all alarm statuses over time:

GET ws/AlarmStatusHistory

This will return a list of all of the alarm statuses for all of your configured alarms, over time.

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
  <resultSize>6</resultSize>
  <requestedSize>1000</requestedSize>
  <pageCursor>38a0c4d9-c45a-11e1-bff1-40409fabfe0f</pageCursor>
  <requestedStartTime>-1</requestedStartTime>
  <requestedEndTime>-1</requestedEndTime>
  <AlarmStatusHistory>
    <id>
      <almId>126</almId>
      <almSourceEntityId>00:24:46:00:06:70:A0</almSourceEntityId>
    </id>
    <almsStatus>0</almsStatus>
    <almsTopic>Alarm.XBeeNodeOffline</almsTopic>
    <cstId>2</cstId>
    <almsUpdateTime>2012-06-26 18:50:38.413</almsUpdateTime>
    <almsPayload>
      <Payload>
        <XbeeCore>
          <xpExtAddr>00:24:46:00:06:70:A0</xpExtAddr>
          <devConnectwareId>00000000-00000000-00409DFF-FF441634</devConnectwareId>
          <cstId>2</cstId>
          <grpId>2</grpId>
          <xpNetAddr>31028</xpNetAddr>
          <xpNodeType>1</xpNodeType>
          <xpMfgId>4250</xpMfgId>
          <xpDiscoveryIndex>1</xpDiscoveryIndex>
          <xpStatus>1</xpStatus>
          <xpUpdateTime>2012-06-26T18:50:38.001Z</xpUpdateTime>
          <grpPath>/CUS0000001_Digi_International</grpPath>
        </XbeeCore>
      </Payload>
    </almsPayload>
  </AlarmStatusHistory>
  <AlarmStatusHistory>
    <id>
      <almId>126</almId>
      <almSourceEntityId>00:24:46:00:06:70:A0</almSourceEntityId>
    </id>
    <almsStatus>0</almsStatus>
    <almsTopic>Alarm.XBeeNodeOffline</almsTopic>
    <cstId>2</cstId>
    <almsUpdateTime>2012-06-26 18:50:38.353</almsUpdateTime>
    <almsPayload>
      <Payload>
        <XbeeCore>
          <xpExtAddr>00:24:46:00:06:70:A0</xpExtAddr>
          <devConnectwareId>00000000-00000000-00409DFF-FF441634</devConnectwareId>
          <cstId>2</cstId>
          <grpId>2</grpId>
          <xpNetAddr>31028</xpNetAddr>
          <xpNodeType>1</xpNodeType>
          <xpMfgId>4250</xpMfgId>
          <xpDiscoveryIndex>1</xpDiscoveryIndex>
          <xpStatus>1</xpStatus>
          <xpUpdateTime>2012-06-26T18:50:38.001Z</xpUpdateTime>
          <grpPath>/CUS0000001_Digi_International</grpPath>
        </XbeeCore>
      </Payload>
    </almsPayload>
  </AlarmStatusHistory>
  <AlarmStatusHistory>
    <id>
      <almId>156</almId>
      <almSourceEntityId>00000000-00000000-00409DFF-FF441634</almSourceEntityId>
    </id>
    <almsStatus>1</almsStatus>
    <almsTopic>Alarm.DeviceOffline</almsTopic>
    <cstId>2</cstId>
    <almsUpdateTime>2012-07-02 15:16:57.385</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>
<dpCapabilities>2618</dpCapabilities>
<grpPath>/CUS0000001_Digi_International</grpPath>
</DeviceCore>
</Payload>
</almsPayload>
</AlarmStatusHistory>
<AlarmStatusHistory>
  <id>
    <almId>157</almId>
    <almsSourceEntityId>00000000-00000000-00409DFF-FF441634</almsSourceEntityId>
  </id>
  <almsStatus>1</almsStatus>
  <almsTopic>Alarm.DeviceOffline</almsTopic>
  <cstId>2</cstId>
  <almsUpdateTime>2012-07-02 15:16:57.47</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>
        <dpCapabilities>2618</dpCapabilities>
        <grpPath>/CUS0000001_Digi_International</grpPath>
      </DeviceCore>
    </Payload>
  </almsPayload>
</AlarmStatusHistory>
</AlarmStatusHistory>
  <id>
    <almId>142</almId>
    <almsSourceEntityId>00000000-00000000-00409DFF-FF4A3946</almsSourceEntityId>
  </id>
  <almsStatus>1</almsStatus>
  <almsTopic>Alarm.DeviceExcessiveConnect</almsTopic>
  <cstId>2</cstId>
  <almsUpdateTime>2012-07-02 15:20:38.386</almsUpdateTime>
  <almsPayload>
    <Payload>
      <DeviceCore>
        <id>
          <devId>32846</devId>
          <devVersion>0</devVersion>
        </id>
        <devRecordStartDate>2012-06-29T13:08:00.000Z</devRecordStartDate>
        <devMac>00:40:9D:4A:39:46</devMac>
        <devCellularModemId>357975020409993</devCellularModemId>
        <devConnectwareId>00000000-00000000-00409DFF-FF4A3946</devConnectwareId>
        <cstId>2</cstId>
        <grpId>2</grpId>
        <devEffectiveStartDate>2012-05-03T20:34:00.000Z</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>0</dpConnectionStatus>
        <dpLastConnectTime>2012-07-02T15:20:09.050Z</dpLastConnectTime>
        <dpContact />
```

```

        <dpDescription>Tech Pubs X4</dpDescription>
        <dpLocation />
        <dpPanId>0xd367</dpPanId>
        <xpExtAddr>00:13:A2:00:40:66:A1:B2</xpExtAddr>
        <dpServerId />
        <dpZigbeeCapabilities>383</dpZigbeeCapabilities>
        <dpCapabilities>2578</dpCapabilities>
        <grpPath>/CUS0000001_Digi_International</grpPath>
    </DeviceCore>
</Payload>
</almsPayload>
</AlarmStatusHistory>
<AlarmStatusHistory>
    <id>
        <alarmId>151</alarmId>
        <almsSourceEntityId>00000000-00000000-00409DFF-FF441634</almsSourceEntityId>
    </id>
    <almsStatus>1</almsStatus>
    <almsTopic>Alarm.DeviceOffline</almsTopic>
    <cstId>2</cstId>
    <almsUpdateTime>2012-07-02 15:25:57.386</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>
                <dpCapabilities>2618</dpCapabilities>
                <grpPath>/CUS0000001_Digi_International</grpPath>
            </DeviceCore>
        </Payload>
    </almsPayload>
</AlarmStatusHistory>
</result>

```

Example #2:

To get the alarm status history details of a specific alarm ID:

GET ws/AlarmStatusHistory/{almId}

17. Carrier Integration

NOTE: Before you can utilize the features described within this chapter and view usage data, you must configure a carrier account within Device Cloud. See the “Enter Carrier Credentials” section of the *Device Cloud User's Guide* (available within the Documentation tab) for instructions on configuring a carrier account.

Also see the “Carrier Account Management page overview” section of the *Device Cloud User's Guide* for detailed information regarding the carrier integration feature and its functionality.

17.1 Introduction

Device Cloud is integrated with various cellular service providers so that you may monitor your actual cellular data usage, according to your provider, directly within the Device Manager user interface. Device Cloud creates a link between your Device Cloud-enabled device and its unique mobile identifier (ICCID for GSM and MEID or ESN for CDMA). Carrier Subscription Management is provisioned for all users as part of their standard Device Manager account subscription. If you are not subscribed to Device Manager you will need to add this service to your account in order to utilize the carrier integration feature. Device Cloud currently supports the following carriers:

- AT&T Control Center
- Deutsche Telekom
- Rogers
- Telefonica
- Verizon Management Center
- Vodafone

Using web services you can query cellular data usage by time ranges, roll-up intervals and perform basic aggregates.

17.2 CarrierAuth

The CarrierAuth Web Service is used to set carrier account credentials or get information regarding which carrier accounts you have configured (without returning password information).

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

HTTP operations supported:

- GET: retrieves a list of your configured carrier accounts
- POST: creates a carrier account
- PUT: updates a carrier account
- DELETE: deletes a carrier account

Elements include:

- cald - ID associated with a specific carrier authentication; a unique identifier will be returned for each CarrierAuth request.
- cstId - Device Cloud ID of customer owning the gateway for which usage information is being gathered (not required for POST or PUT).
- prvName - cellular service provider name abbreviated as: ATT, DeutscheTelekom, Rogers, Telefonica, Verizon or Vodafone.
- caUserName - the username associated with your carrier account. This is the username that was provided by your business account carrier which you used to setup your carrier account within Device Cloud.
- caPassword - cellular service provider password. This is the password that was provided by your business account carrier.
- caUpdateTime - date and time (ISO 8601 format) when your carrier account information was last updated
- caLicenseKey1 is required for AT&T, Rogers, Telefonica & Deutsche Telekom which is the Admin license key
- caLicenseKey2 is required for Deutsche Telekom which is the Monitor license key

Examples:

GET

To get a list of your configured carrier accounts (without returning password information):

GET /ws/CarrierAuth

This will return a list of the carrier accounts that are configured within your Device Cloud account. The following example contains two carrier accounts, one for AT&T and one for Vodafone.

```
<?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>
```

POST

To set carrier account credentials:

POST /ws/CarrierAuth

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

Where:

- prvName should be set to ATT, DeutscheTelekom, Rogers, Telefonica, Verizon, or Vodafone (depending on your cellular service provider)
- caUserName and caPassword are required for all carriers. The values for these fields will be provided by your business account carrier.
- caLicenseKey1 is required for AT&T, Deutsche Telekom, Rogers & Telefonica which is the Admin license key
- caLicenseKey2 is required for Deutsche Telekom which is the Monitor license key

This will create a carrier account (AT&T in this case) within Device Cloud.

PUT

To update a carrier account:

PUT /ws/CarrierAuth

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

This will update the call within example 2 to include a cald element.

DELETE

To delete a carrier account:

DELETE ws/CarrierAuth/{subscription_id}

Will delete the carrier account matching the specified subscription ID.

17.3 CarrierSubscription

Device Cloud calculates a daily delta value for how much usage has changed for each of your carrier integrated devices. These deltas are stored as values in carrier subscriptions. The CarrierSubscription Web Service is used to retrieve this carrier subscription usage information on a per-device basis.

URI: http://<hostname>/ws/CarrierSubscription

HTTP operations supported:

- GET: query per-device carrier subscription usage information

Elements include:

- cstId - Device Cloud ID of customer owning the gateway for which usage information is being gathered
- id - type of usage information being gathered, either data (transferred or sessions) or SMS
- units - units the usage information is reported in (user defined)

Examples:

Example #1:

To get a list of all the carrier subscriptions associated with your Device Cloud account:

GET ws/CarrierSubscription

This will return.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
  <resultSize>3</resultSize>
  <requestedSize>1000</requestedSize>
  <pageCursor>59cf54bf-3-9278ff4f</pageCursor>
  <CarrierSubscription>
    <cstId>3</cstId>
    <id>DATA/89011704258000326078/sessions</id>    <!-- carrier subscription #1 -->
    <units>count</units>
  </CarrierSubscription>
  <CarrierSubscription>
    <cstId>3</cstId>
    <id>DATA/89011704258000326078/transferred</id>    <!-- carrier subscription #2 -->
    <units>kbytes</units>
  </CarrierSubscription>
  <CarrierSubscription>
    <cstId>3</cstId>
    <id>SMS/89011704258000326078/messages</id>    <!-- carrier subscription #3 -->
    <units>count</units>
  </CarrierSubscription>
</result>
```

As you can see, three carrier subscriptions were returned: one for data sessions, one for data transferred, and one for SMS messages. The 89011704258000326078 number displayed within each of these carrier subscriptions corresponds with the ICCID, MEID, or ESN number (whichever is appropriate) of the device for which usage information is being gathered.

Example #2

You can further examine one of the carrier subscription entries returned in the previous call by issuing the following request:

GET ws/CarrierSubscription/{subscription_id}

For example, to get a more in-depth look at the data transferred carrier subscription (carrier subscription #2 above):

GET ws/CarrierSubscription/DATA/89011704258000326078/transferred

Returns:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
  <resultSize>1</resultSize>
  <requestedSize>1000</requestedSize>
  <pageCursor>26b989fa-1-d826bb9c</pageCursor>
  <CarrierSubscription>
    <cstId>3</cstId>
    <id>DATA/89011704258000326078/transferred</id>
    <units>kbytes</units>
  </CarrierSubscription>
</result>
```

17.4 CarrierUsage

The CarrierUsage Web Service is used to get usage information for a specific carrier subscription. This web service must be used in conjunction with the CarrierSubscription Web Service, as a subscription ID must be included in the web services request.

URI: http://<hostname>/ws/CarrierUsage{subscription_id}

HTTP operations supported:

- GET: query carrier usage information for a specific carrier subscription

Elements include:

- cstId - Device Cloud ID of customer owning the gateway for which usage information is being gathered
- id - carrier subscription for which the gathered usage information was obtained
- timestamp - [timestamp](#) for when the usage data was created
- data - numerical value of the actual amount of usage data

NOTE: Each data usage value is a delta over the previous day. Every day Device Cloud calculates the total usage amount for the past day. This total then becomes a new carrier usage value.

Examples:

Example #1:

To get a list of carrier usage information for a specific carrier subscription:

GET ws/CarrierUsage/{subscription_id}

Note: A subscription ID must be included with the request. To learn how to obtain this subscription ID, refer to the [Carrier Subscription](#) section. For the examples within this section we are using a data transferred carrier subscription.

For example, GET ws/CarrierUsage/DATA/89011704258000326078/transferred, would return:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
  <resultSize>744</resultSize>
  <requestedSize>1000</requestedSize>
  <pageCursor>bafafe29-2a1f-11e2-bde8-4040690fdd40</pageCursor>
  <requestedStartTime>-1</requestedStartTime>
  <requestedEndTime>-1</requestedEndTime>
  <CarrierUsage>
    <cstId>3</cstId>
    <id>DATA/89011704258000326078/transferred</id>
    <timestamp>1287999906641</timestamp>
    <data>4.0</data>
  </CarrierUsage>
  <CarrierUsage>
    <cstId>3</cstId>
    <id>DATA/89011704258000326078/transferred</id>
    <timestamp>1288086306641</timestamp>
```

```
<data>2.0</data>
</CarrierUsage>
<id>DATA/89011704258000326078/transferred</id>
<timestamp>1352129673379</timestamp>
<data>17.41</data>
</CarrierUsage>
<CarrierUsage>
  <csId>3</csId>
  <id>DATA/89011704258000326078/transferred</id>
  <timestamp>1352171365235</timestamp>
  <data>3.07</data>
</CarrierUsage>
<CarrierUsage>
  <csId>3</csId>
  <id>DATA/89011704258000326078/transferred</id>
  <timestamp>1352234341519</timestamp>
  <data>0.0</data>
</CarrierUsage>
<CarrierUsage>
  <csId>3</csId>
  <id>DATA/89011704258000326078/transferred</id>
  <timestamp>1352257467153</timestamp>
  <data>2.0</data>
</CarrierUsage>
<CarrierUsage>
  <csId>3</csId>
  <id>DATA/89011704258000326078/transferred</id>
  <timestamp>1352432654390</timestamp>
  <data>6.0</data>
</CarrierUsage>
</result>
```

NOTE: The 89011704258000326078 number displayed within each carrier usage entry corresponds with the ICCID, MEID, or ESN number (whichever is appropriate) of the device for which carrier subscription information is being gathered.

You can also perform various roll-up requests on these usages using any of the supported roll-up methods. See the [Data Points](#) section for details on the supported roll-up methods, and for additional roll-up related web services examples.

Example #2:

To get weekly usage data from a carrier subscription:

GET /ws/CarrierUsage/DATA/89011704258000326078/transferred?rollupInterval=week&rollupMethod=average

Returns a roll-up containing weekly average usage data transferred for the specified carrier subscription.

Example #3: To get a daily sum of usage data from a carrier subscription:

GET /ws/CarrierUsage/DATA/89011704258000326078/transferred?rollupMethod=sum&rollupInterval=day

Returns a daily roll-up of the sum of all usage data transferred for the specified carrier subscription.

17.5 NetworkInterfaceSubscriptionCore

The NetworkInterfaceSubscriptionCore Web Service is used to get subscription information for your devices based upon the [NetworkInterface](#) records.

This indicates whether carrier accounts have been assigned to the NetworkInterface records and what metrics will be collected for each.

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

HTTP operations supported:

- GET: retrieve NetworkInterface level subscription information

Elements include:

- **csId** - Device Cloud ID of customer owning the gateway for which usage information is being gathered
- **id**
 - **nild** - NetworkInterface id
 - **subId** - Subscription id (0 if not assigned to any carrier)

Examples:

Example #1:

To get a list of NetworkInterface level subscription information:

GET ws/ NetworkInterfaceSubscriptionCore

This will return a list similar to the following. In the example return below, three NetworkInterface level entries were returned:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
  <resultTotalRows>3</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>3</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <NetworkInterfaceSubscriptionCore>
    <id>
      <niId>3</niId>
      <subId>147</subId>
    </id>
    <niInterfaceType>1</niInterfaceType>
    <niSimId>89011704258000326078</niSimId>
    <niStatus>active</niStatus>
    <niPhone>15071234567</niPhone>
    <niActivePhone>false</niActivePhone>
    <devId>5</devId>
    <csiId>3</csiId>
    <grpId>3</grpId>
    <devMac>00:40:9D:44:19:E2</devMac>
    <devCellularModemId>354976040020193</devCellularModemId>
    <devConnectwareId>00000000-00000000-00409DFF-FF4419E2</devConnectwareId>
    <subStatus>0</subStatus>
    <subStatusLocked>false</subStatusLocked>
    <subCollectionLevel>2</subCollectionLevel>
    <svcId>17</svcId>
    <rpId>26</rpId>
  </NetworkInterfaceSubscriptionCore>
</-- entry #1 -->

```

```

<svcName>ATT Data</svcName>
<svcDescription>ATT Cellular Data</svcDescription>
<svcMetricCount>2</svcMetricCount>
<svcMetric>transferred</svcMetric>
<svcMetric2>sessions</svcMetric2>
<svcUnits>kbytes</svcUnits>
<svcUnits2>count</svcUnits2>
<prvId>12</prvId>
<svcNumber>0</svcNumber>
<svcLevel>3</svcLevel>
<svcReqInterfaceType>1</svcReqInterfaceType>
<prvName>ATT</prvName>
<prvDescription>ATT Cellular</prvDescription>
</NetworkInterfaceSubscriptionCore>
<NetworkInterfaceSubscriptionCore>                                <!-- entry #2 -->
  <id>
    <niId>3</niId>
    <subId>148</subId>
  </id>
  <niInterfaceType>1</niInterfaceType>
  <niSimId>89011704258000326078</niSimId>
  <niStatus>active</niStatus>
  <niPhone>15071234567</niPhone>
  <niActivePhone>false</niActivePhone>
  <devId>5</devId>
  <cstId>3</cstId>
  <grpId>3</grpId>
  <devMac>00:40:9D:44:19:E2</devMac>
  <devCellularModemId>354976040020193</devCellularModemId>
  <devConnectwareId>00000000-00000000-00409DFF-FF4419E2</devConnectwareId>
  <subStatus>0</subStatus>
  <subStatusLocked>false</subStatusLocked>
  <subCollectionLevel>2</subCollectionLevel>
  <svcId>18</svcId>
  <rpId>27</rpId>
  <svcName>ATT SMS</svcName>
  <svcDescription>ATT Cellular SMS</svcDescription>
  <svcMetricCount>1</svcMetricCount>
  <svcMetric>messages</svcMetric>
  <svcUnits>count</svcUnits>
  <prvId>12</prvId>
  <svcNumber>1</svcNumber>
  <svcLevel>3</svcLevel>
  <svcReqInterfaceType>1</svcReqInterfaceType>
  <prvName>ATT</prvName>
  <prvDescription>ATT Cellular</prvDescription>
</NetworkInterfaceSubscriptionCore>
<NetworkInterfaceSubscriptionCore>                                <!-- entry #3 -->
  <id>
    <niId>4</niId>
    <subId>0</subId>
  </id>
  <niInterfaceType>1</niInterfaceType>
  <niSimId>8901410221226919394</niSimId>
  <niImsi>310410226919394</niImsi>
  <niPhone>17143306338</niPhone>
  <niActivePhone>true</niActivePhone>
  <niIdigiPhone>32075</niIdigiPhone>
  <niIdigiServiceId>idgg</niIdigiServiceId>
  <devId>2</devId>
  <cstId>3</cstId>
  <grpId>3</grpId>
  <devMac>00:40:9D:49:2F:12</devMac>
  <devCellularModemId>353229021304330</devCellularModemId>
  <devConnectwareId>00000000-00000000-00409DFF-FF492F12</devConnectwareId>
</NetworkInterfaceSubscriptionCore>
</result>

```

18. Data Streams

18.1 Introduction

The Data Streams API allows data to be stored in a time series for long periods of time. Data streams represent time series data, a sequence of data points. You can create a data stream with web services by uploading data points, or using the DIA or Smart Energy frameworks with Device Cloud. You can query the data by time ranges, roll-up intervals and perform basic aggregates.

Time series data involves two concepts: data points and data streams.

- **Data points** are the individual values which are stored at specific times.
- **Data streams** are containers of data points. Data streams hold meta data about the data points held within them. The data streams and the data points they hold are addressed using hierarchical paths (much like folders).

18.2 Data Streams

Data streams contain the meta data for time series data. They contain information about the data points as well as other contained streams.

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

HTTP operations supported:

- GET: query one or more data streams
- POST: create a data stream (supports multiple data streams in a single call)
- PUT: create or update a data stream
- DELETE: delete existing data streams

Elements include:

- cstId - customer ID of customer that owns the data stream
- streamId - the stream ID (i.e. stream path)
- dataType - type of data stored in this data stream. Valid types include:
 - 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 - units the data is reported in (user defined)
- description - description of the data stream
- forwardTo - comma separated list of streams to replicate data points to
- dataTtl - the 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 - the 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 (this is not writeable in PUT or POST requests)
 - id - data point ID
 - timestamp - data point client `timestamp`
 - serverTimestamp - `timestamp` when data point received by the server
 - data - the data value of the data point
 - description - description of the data point
 - quality - the quality of the data point (user-defined 32-bit integer value)
 - location - each data point can have geo-location information associated with it which indicates the location when the data point was recorded. Geo-location is represented as a comma separated list of floats in order of lat, long, elevation.

GET

GET is used to retrieve a list of data streams, or a specific stream by its ID. You can query by the full path of a stream or retrieve all children of a "parent". For example if the stream's parent/childA, parent/ChildB exist you can retrieve both with a query to `/ws/DataStream/parent` or to a specific one with `/ws/DataStream/parent/childA`.

Query parameters:

- size - maximum number of data streams to return
- pageCursor - page cursor returned from a previous request that can be used to retrieve the next page of data

Example #1:

To get a list of all data streams:

GET `/ws/DataStream`

Returns a list of data streams.

Example #2:

To get information about a specific data stream using its stream ID:

GET `/ws/DataStream/{streamId}`

Returns details for the specific data stream associated with the specified stream ID.

POST

POST allows you to create a new data stream using the elements described previously.

Example:

POST `/ws/DataStream/{<stream-path>}`

```
<DataStream>
  <streamId>my/stream2</streamId>    <!-- enter a name for your data stream -->
  <dataType>INTEGER</dataType>      <!-- this can be any of the supported dataTypes -->
  <units>grams</units>               <!-- this can be any unit of your choosing -->
  <dataTtl>86400</dataTtl>           <!-- one day expressed in seconds -->
  <rollupTtl>63072000</rollupTtl>    <!-- two years expressed in seconds -->
</DataStream>
```

Using POST also allows a list of multiple data streams in a single request (wrapped in a `<list>` element).

PUT

PUT allows you to update an existing data stream.

Example:

For example, to modify the `dataType` field of an existing data stream from `LONG` to `INTEGER` and add a description:

PUT `/ws/DataStream/{<stream-path>}`

```
<DataStream>
  <dataType>INTEGER</dataType>
  <description>My thermostat</description>
</DataStream>
```

NOTE: All the attributes described in the POST section can also be specified for PUT.

DELETE

DELETE is used to delete a data stream, including all of its data points and roll-ups.

Example:

To delete a data stream:

DELETE `/ws/DataStream/{<stream-path>}`

Will delete the data stream matching the specified stream path.

18.2.1 Replicating and Joining Data Streams

Data streams typically hold data points for a specific attribute on a device; for example, the temperature from a specific thermostat. However, there are times where you may want to query for data across multiple data streams; for example, temperature readings for all your thermostats. This can be done in 2 ways:

1. **join:** join (or merge) the data points from 2 or more data streams when querying.
2. **replication:** when writing a data point to a data stream, write the data point to one or more other data streams.

18.2.1.1 Joining Data Streams

Joining data streams allows you to query data points from 2 or more data streams without actually duplicating the data in storage. This provides a quick and easy way to query existing streams. This will not scale to very many data streams. Also, you cannot do roll-ups on joined data streams.

18.2.1.2 Replicating Data Streams

Replicating data streams allows you to create new data streams that have a copy of the data points from one or more other data streams. This provides much faster queries and allows the data to be rolled up and aggregated (if all the data points are of the same numeric type).

18.3 Data Points

Data points are the individual data values stored at specific times and are stored in data streams. All data points have a [timestamp](#) and a value. (Actually they have 2 timestamps: the timestamp assigned by the client and a timestamp when the data point was received on the server.) Data points can also have additional meta data associated with them such as a description, geo-position, etc. When writing data points, if the data stream doesn't exist it will be created automatically.

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

HTTP operations supported:

- GET: retrieve data points for a data stream
- POST: create a data point (supports multiple data points in a single call)
- PUT: not supported, you can't update existing data points
- DELETE: delete existing data points

Elements include:

- id - the ID of the data point
- csId - customer ID of customer that owns the data point
- streamId - the stream ID (i.e. stream path) the data point was initially written to. Typically this is the data stream that the data point belongs to, but if using replication (forwardTo) it may be different.
- timestamp - client assigned [timestamp](#) (if not specified by the client then this will be the same as serverTimestamp)
- serverTimestamp - [timestamp](#) when the data point was stored on the server. Not writable by the client.
- data - the actual data value
- description - description of this data point
- quality - user-defined 32-bit integer value representing the quality of the data in the data point
- location - each data point can have geo-location information associated with it which indicates the location when the data point was recorded. Geo-location is represented as a comma separated list of floats in order of lat, long, elevation (degrees, degrees, meters).

In addition to the above fields, you can also specify some data stream fields when inserting data points. If data stream fields are specified then the data stream will be updated with those values. Since the data stream will be created if it does not yet exist, this allows the data stream to be initialized without having to create it explicitly. These fields include the following (refer to the [Data Streams](#) section for more details on these fields):

- dataType - type of data stored in this data stream
- units - units the data is reported in (user defined)
- forwardTo - comma separated list of streams to replicate data points to

GET

GET is used to retrieve data points for a data stream. The stream path is required in the URL.

Query parameters:

- size - maximum number of data streams to return
- pageCursor - page cursor returned from a previous request that can be used to retrieve the next page of data
- startTime - [timestamp](#) for the start time (inclusive)
- endTime - [timestamp](#) for the end time (exclusive)
- timeline - which timestamps (client or server) to use in the request (default is client).

NOTE: Roll-ups are only supported for the client timeline (see [rollupInterval](#), [rollupMethod](#), and [timezone](#)). A roll-up is a combination across a set interval of data points by some aggregation function (for example, the hourly sum of all sodas purchased at a vending machine). See the "rollupMethod" element for supported aggregations.

- order - ascending or descending (actually checks if parameter value starts with 'asc' or 'desc')
- rollupInterval - the roll-up interval. Valid intervals include:
 - half (roll-up values in half hour intervals)
 - hour (roll-up values in hourly intervals)
 - day (roll-up values in daily intervals)
 - week (roll-up values in weekly intervals)
 - month (roll-up values in monthly intervals)
- rollupMethod - the aggregation method applied to values in the roll-up intervals. All aggregation methods are represented as doubles. Valid aggregations include:
 - sum (sum of all values in each roll-up interval)
 - average (average of all values in each roll-up interval)
 - min (minimum value in each roll-up interval)
 - max (maximum value in each roll-up interval)
 - count (count of all data points in each roll-up interval)
 - standarddev (standard deviation of all values in each roll-up interval)
- timezone: timezone to calculate roll-ups in. This determines roll-up interval boundaries and only applies to roll-ups of a day or larger (e.g. day, week, month). Note that it does *not* apply to the startTime and endTime parameters. See the [Timestamps](#) and [Supported time zones](#) sections for more information.

Example #1:

To get a list of all data points for a specific data stream:

GET /ws/DataPoint[<stream-path>]

Returns a list of all of the data points pertaining to the specified data stream.

Example #2:

To get hourly data from a data stream:

GET /ws/DataPoint/device1/temp?rollupInterval=hour&rollupMethod=average

Returns hourly average temperature data for the specified data stream.

Example #3:

To get a daily sum of all data points:

GET ws/DataPoint/dia/channel/00000000-00000000-12300000-00000000A/sensor0/temperature?rollupMethod=sum&rollupInterval=day

Returns a daily roll-up of the sum of all data values in the specified data stream.

POST

POST allows you to create one or more data points within an existing data stream using the elements described previously.

Example: POST /ws/DataStream

```
<DataPoint>
  <data>42</data>
  <!-- Everything below this is optional -->

  <streamId>device1/temp</streamId>
  <description>Temperature at device 1</description>
  <location>0.0, 0.0, 0.0</location>
  <!-- lat, long, elevation -->
  <quality>99</quality>
  <!-- <timestamp>[epoch long or ISO8601, if not set defaults to now]>/timestamp-->

  <!-- the following sets the data stream the data point belongs to (optional)-->
```

```
<dataType>float</dataType>
<units>Kelvin</units>
<forwardTo>allDevices/temp, allDevices/metrics</forwardTo>
</DataPoint>
```

This will create a data point with a value of 42.

DELETE

DELETE is used to delete data points for a specific data point ID or for time ranges. The time ranges can be either client or server [timestamps](#).

Example #1:

To delete a data point by ID:

```
DELETE ws/DataPoint/device1/temp/9b55dfc7-d10a-11e1-b864-000c29d68277
```

Will delete the data point matching the 9b55dfc7-d10a-11e1-b864-000c29d68277 ID.

Example #2:

To delete data points for a time range:

```
DELETE ws/DataPoint/device1/temp?startTime=2012-07-18T12:00:00.000Z&endTime=2012-07-18T12:30:00.000Z
```

Will delete all data points within the specified range.

18.3.1 Timestamps

When working with data points, all timestamps can be specified as epoch time (number of milliseconds from midnight January 1, 1970 UTC) or ISO 8601 format. All timestamps are returned as epoch time. If a timestamp is specified in ISO 8601 format but without a time zone designator/offset then UTC will be used. Note that the *timezone* parameter is not used in timestamps. It is only used to determine interval boundaries for roll-up calculations.

The data point timestamps are typically client (or device) generated timestamps. However, data points will also contain server timestamps. You can query by either the client or server timestamps by specifying the timeline parameter. Querying by the server timestamps allows you to determine when new data comes in, even if it has an old timestamp.

ISO 8601 is the International Standard for the representation of dates and times. Please refer to the [ISO 8601 Date & Duration Reference](#) Appendix for more information on expressing dates in ISO 8601 format.

18.4 Viewing time series data roll-ups

Once you have created or edited a data stream and added data points to that stream you can see the aggregate of that data within Device Cloud. To view various roll-ups of your data see the "Data Streams" Chapter of the *Device Cloud User's Guide* (available within the Documentation tab).

18.5 Inserting data from devices

There are multiple mechanisms for inserting data points into data streams. Data points can be inserted with the same interface that is used when reading with web services (see /ws/DataPoint above). However, this method can be problematic as it requires the use of customer account credentials. To upload data from devices you can use existing mechanisms that instead use device tokens and authenticate at the device level. This includes the DIA and Smart Energy frameworks along with any request (TCP, UDP, SMS, Satellite) that uploads data to a file prefixed with the "DataPoint" path. For details on each transport see the device implementations.

18.5.1 DIA and Smart Energy

By default, all data uploaded to the DIA or Smart Energy frameworks gets cloned into data streams but the format is slightly different for each.

Smart Energy

Attribute data uploaded to the Smart Energy framework will get copied into a data stream matching the following path:

```
se/attr/<Device Id>/<Xbee Address>/<Endpoint Id>/<Cluster Type>/<Cluster Id>/<Attribute Id>
```

For example, a device with a Device ID of "00000000-00000000-00000000-00000000" and an Xbee node with an address of "CA:FE:CA:FE:CA:FE:CA" would upload data from an endpoint of 18, cluster type of 0, cluster id of 513, and would upload to an attribute with an id of 0. The data will be included in the data stream:

```
se/attr/00000000-00000000-00000000-00000000/CA:FE:CA:FE:CA:FE:CA/18/0/513/0
```

Event data will be the same as above but instead of "attr" as the second element in the path, it would be "event".

DIA

Channel data uploaded to the DIA framework will be copied into a data stream matching the following path:

```
dia/channel/<Device Id>/<instance>/<channel>
```

For example: dia/channel/00000000-00000000-12300000-00000009/sensor0/temperature

Access

To get a list of all attributes for a device you can query `/ws/DataStream/se/attr/Device/d` which will provide a list of all data streams matching that prefix. If you want, for example, the average of an attribute for all of your devices, you can set each data stream to forward to a common stream (i.e. `se/attr/all/xbee/endpoint/cluster/clusterid/attribute`).

18.5.2 Direct Device Uploads

Devices can upload directly to data streams. This can be achieved over any of the existing transports (TCP, UDP, SMS, and Satellite). The path specified in the data service message will start with "DataPoint" and the rest will be mapped to a data stream appended to the Device ID and group path. For example, if the device sends a data point file specifying the filename "DataPoint/temp1", the data point will be added to the data stream `<device-id>/temp1`. The file *must* follow one of the expected formats and it *must* include the format it's using in the upload; specifying the format is done via the 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.

XML

The XML format reuses the same format used in /ws/DataPoint PUT. A limitation is added where 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.

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

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).

Binary Concise Alternative Format

The disadvantage to using the XML format is that it is very verbose. This binary alternative format can be used in order to be more concise. You may simply specify a simple value instead of XML or CSV data. When this value is pushed to /DataPoint, it will be stored in complete as is in time-series data (in the exact binary format as provided). For details on endianness, bit lengths, etc for supported data types see the [Data Streams](#) section. However, data types are not required. Data can be 1 byte status indicators or 10k images but Device Cloud 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

18.6 Supported time zones

Time zones can be specified as standard offsets from GMT (e.g. -06:00 for US Central time zone) or by their canonical ID. The following are valid canonical IDs for time zones:

Supported Time Zones - Africa			
Africa/Abidjan	Africa/Accra	Africa/Addis_Ababa	Africa/Algiers
Africa/Asmara	Africa/Asmera	Africa/Bamako	Africa/Bangui
Africa/Banjul	Africa/Bissau	Africa/Blantyre	Africa/Brazzaville
Africa/Bujumbura	Africa/Cairo	Africa/Casablanca	Africa/Ceuta
Africa/Conakry	Africa/Dakar	Africa/Dar_es_Salaam	Africa/Djibouti
Africa/Djibouti	Africa/Douala	Africa/El_Aaiun	Africa/Freetown
Africa/Gaborone	Africa/Harare	Africa/Johannesburg	Africa/Kampala
Africa/Khartoum	Africa/Kigali	Africa/Kinshasa	Africa/Lagos
Africa/Libreville	Africa/Lome	Africa/Luanda	Africa/Lubumbashi
Africa/Lusaka	Africa/Malabo	Africa/Maputo	Africa/Maseru
Africa/Mbabane	Africa/Mogadishu	Africa/Monrovia	Africa/Nairobi
Africa/Ndjamena	Africa/Niamey	Africa/Nouakchott	Africa/Ouagadougou
Africa/Porto-Novo	Africa/Sao_Tome	Africa/Timbuktu	Africa/Tripoli
Africa/Tunis	Africa/Windhoek		
Supported Time Zones - America			
America/Adak	America/Anchorage	America/Anguilla	America/Antigua
America/Araguaina	America/Argentina/Buenos_Aires	America/Argentina/Catamarca	America/Argentina/ComodRivadavia
America/Argentina/Cordoba	America/Argentina/Jujuy	America/Argentina/La_Rioja	America/Argentina/Mendoza
America/Argentina/Rio_Gallegos	America/Argentina/Salta	America/Argentina/San_Juan	America/Argentina/San_Luis
America/Argentina/Tucuman	America/Argentina/Ushuaia	America/Aruba	America/Asuncion
America/Atikokan	America/Atka	America/Bahia	America/Bahia_Banderas
America/Barbados	America/Belem	America/Belize	America/Blanc-Sablon
America/Boa_Vista	America/Bogota	America/Boise	America/Buenos_Aires
America/Cambridge_Bay	America/Campo_Grande	America/Cancun	America/Caracas

America/Catamarca	America/Cayenne	America/Cayman	America/Chicago
America/Chihuahua	America/Coral_Harbour	America/Cordoba	America/Costa_Rica
America/Cuiaba	America/Curacao	America/Danmarkshavn	America/Dawson
America/Dawson_Creek	America/Denver	America/Detroit	America/Dominica
America/Edmonton	America/Eirunepe	America/El_Salvador	America/Ensenada
America/Fort_Wayne	America/Fortaleza	America/Glace_Bay	America/Godthab
America/Goose_Bay	America/Grand_Turk	America/Grenada	America/Guadeloupe
America/Guatemala	America/Guayaquil	America/Guyana	America/Halifax
America/Havana	America/Hermosillo	America/Indiana/Indianapolis	America/Indiana/Knox
America/Indiana/Marengo	America/Indiana/Petersburg	America/Indiana/Tell_City	America/Indiana/Vevay
America/Indiana/Vincennes	America/Indiana/Winamac	America/Indianapolis	America/Inuvik
America/Iqaluit	America/Jamaica	America/Jujuy	America/Juneau
America/Kentucky/Louisville	America/Kentucky/Monticello	America/Knox_IN	America/Kralendijk
America/La_Paz	America/Lima	America/Los_Angeles	America/Louisville
America/Lower_Princes	America/Maceio	America/Managua	America/Manaus
America/Marigot	America/Martinique	America/Matamoros	America/Mazatlan
America/Menominee	America/Menominee	America/Merida	America/Metlakatla
America/Mexico_City	America/Miquelon	America/Moncton	America/Monterrey
America/Montevideo	America/Montreal	America/Montserrat	America/Nassau
America/New_York	America/Nipigon	America/Nome	America/Noronha
America/North_Dakota/Beulah	America/North_Dakota/Center	America/North_Dakota/New_Salem	America/Ojinaga
America/Panama	America/Pangnirtung	America/Paramaribo	America/Phoenix
America/Port-au-Prince	America/Port_of_Spain	America/Porto_Acre	America/Porto_Velho
America/Puerto_Rico	America/Rainy_River	America/Rankin_Inlet	America/Recife
America/Regina	America/Resolute	America/Rio_Branco	America/Rosario
America/Santa_Isabel	America/Santarem	America/Santiago	America/Santo_Domingo
America/Sao_Paulo	America/Scoresbysund	America/Shiprock	America/Sitka
America/St_Barthelemy	America/St_Johns	America/St_Kitts	America/St_Lucia
America/St_Thomas	America/St_Vincent	America/Swift_Current	America/Tegucigalpa
America/Thule	America/Thunder_Bay	America/Tijuana	America/Toronto
America/Tortola	America/Vancouver	America/Virgin	America/Whitehorse
America/Winnipeg	America/Yakutat	America/Yellowknife	
Supported Time Zones - Antarctica			
Antarctica/Casey	Antarctica/Davis	Antarctica/DumontDUrville	Antarctica/Macquarie
Antarctica/Mawson	Antarctica/McMurdo	Antarctica/Palmer	Antarctica/Rothera
Antarctica/South_Pole	Antarctica/Syowa	Antarctica/Vostok	Arctic/Longyearbyen
Supported Time Zones - Asia			
Asia/Aden	Asia/Almaty	Asia/Amman	Asia/Anadyr
Asia/Aqtai	Asia/Aqtobe	Asia/Ashgabat	Asia/Ashkhabad
Asia/Baghdad	Asia/Bahrain	Asia/Baku	Asia/Bangkok
Asia/Beirut	Asia/Bishkek	Asia/Brunei	Asia/Calcutta
Asia/Choibalsan	Asia/Chongqing	Asia/Chungking	Asia/Colombo
Asia/Dacca	Asia/Damascus	Asia/Dhaka	Asia/Dili
Asia/Dubai	Asia/Dushanbe	Asia/Gaza	Asia/Harbin
Asia/Ho_Chi_Minh	Asia/Hong_Kong	Asia/Hovd	Asia/Irkutsk
Asia/Istanbul	Asia/Jakarta	Asia/Jayapura	Asia/Jerusalem
Asia/Kabul	Asia/Kamchatka	Asia/Karachi	Asia/Kashgar
Asia/Kathmandu	Asia/Katmandu	Asia/Kolkata	Asia/Krasnoyarsk
Asia/Kuala_Lumpu	Asia/Kuching	Asia/Kuwait	Asia/Macao
Asia/Macau	Asia/Magadan	Asia/Makassar	Asia/Manila
Asia/Muscat	Asia/Nicosia	Asia/Novokuznetsk	Asia/Novosibirsk
Asia/Omsk	Asia/Oral	Asia/Phnom_Penh	Asia/Pontianak
Asia/Pyongyang	Asia/Qatar	Asia/Qyzylorda	Asia/Rangoon
Asia/Riyadh	Asia/Saigon	Asia/Sakhalin	Asia/Samarkand
Asia/Seoul	Asia/Seoul	Asia/Singapore	Asia/Taipei
Asia/Tashkent	Asia/Tbilisi	Asia/Tehran	Asia/Tel_Aviv
Asia/Thimbu	Asia/Thimphu	Asia/Tokyo	Asia/Ujung_Pandang
Asia/Ulaanbaatar	Asia/Ulan_Bator	Asia/Urumqi	Asia/Vientiane
Asia/Vladivostok	Asia/Yakutsk	Asia/Yekaterinburg	Asia/Yerevan
Supported Time Zones - Atlantic			
Atlantic/Azores	Atlantic/Bermuda	Atlantic/Canary	Atlantic/Cape_Verde
Atlantic/Faeroe	Atlantic/Faroe	Atlantic/Jan_Mayen	Atlantic/Madeira
Atlantic/Reykjavik	Atlantic/South_Georgia	Atlantic/St_Helena	Atlantic/Stanley
Supported Time Zones - Australia			
Australia/ACT	Australia/Adelaide	Australia/Brisbane	Australia/Broken_Hill
Australia/Canberra	Australia/Currie	Australia/Darwin	Australia/Eucla
Australia/Hobart	Australia/LHI	Australia/Lindeman	Australia/Lord_Howe
Australia/Melbourne	Australia/NSW	Australia/North	Australia/Perth
Australia/Queensland	Australia/South	Australia/Sydney	Australia/Tasmania
Australia/Victoria	Australia/West	Australia/Yancowinna	
Supported Time Zones - Brazil			
Brazil/Acre	Brazil/DeNoronha	Brazil/East	Brazil/West
Supported Time Zones - C			
CET	CST6CDT	Chile/Continental	Chile/EasterIsland
Cuba			
Supported Time Zones - Canada			
Canada/Atlantic	Canada/Central	Canada/East-Saskatchewan	Canada/Eastern
Canada/Mountain	Canada/Newfoundland	Canada/Pacific	Canada/Saskatchewan
Canada/Yukon			
Supported Time Zones - E			
EET	EST	EST5EDT	Egypt
Eire			
Supported Time Zones - Etc			
Etc/GMT	Etc/GMT+0	Etc/GMT+1	Etc/GMT+10
Etc/GMT+11	Etc/GMT+12	Etc/GMT+2	Etc/GMT+3
Etc/GMT+4	Etc/GMT+5	Etc/GMT+6	Etc/GMT+7

Etc/GMT+8	Etc/GMT+9	Etc/GMT-0	Etc/GMT-1
Etc/GMT-10	Etc/GMT-11	Etc/GMT-12	Etc/GMT-13
Etc/GMT-14	Etc/GMT-2	Etc/GMT-3	Etc/GMT-4
Etc/GMT-5	Etc/GMT-6	Etc/GMT-7	Etc/GMT-8
Etc/GMT-9	Etc/GMT0	Etc/Greenwich	Etc/UCT
Etc/UTC	Etc/Universal	Etc/Zulu	
Supported Time Zones - Europe			
Europe/Amsterdam	Europe/Andorra	Europe/Athens	Europe/Belfast
Europe/Belgrade	Europe/Berlin	Europe/Bratislava	Europe/Brussels
Europe/Bucharest	Europe/Budapest	Europe/Chisinau	Europe/Copenhagen
Europe/Dublin	Europe/Gibraltar	Europe/Guernsey	Europe/Helsinki
Europe/Isle_of_Man	Europe/Istanbul	Europe/Jersey	Europe/Kaliningrad
Europe/Kiev	Europe/Lisbon	Europe/Ljubljana	Europe/London
Europe/Luxembourg	Europe/Madrid	Europe/Malta	Europe/Mariehamn
Europe/Minsk	Europe/Monaco	Europe/Moscow	Europe/Nicosia
Europe/Oslo	Europe/Paris	Europe/Podgorica	Europe/Prague
Europe/Riga	Europe/Rome	Europe/Samara	Europe/San_Marino
Europe/Sarajevo	Europe/Simferopol	Europe/Skopje	Europe/Sofia
Europe/Stockholm	Europe/Tallinn	Europe/Tirane	Europe/Tiraspol
Europe/Uzhgorod	Europe/Vaduz	Europe/Vatican	Europe/Vienna
Europe/Vilnius	Europe/Volgograd	Europe/Warsaw	Europe/Zagreb
Europe/Zaporozhye	Europe/Zurich		
Supported Time Zones - G			
GB	GB-Eire	GMT	GMT+0
GMT-0	GMT0	Greenwich	
Supported Time Zones - H			
HST	HongKong		
Supported Time Zones - I			
Iceland	Iran	Israel	
Supported Time Zones - Indian			
Indian/Antananarivo	Indian/Chagos	Indian/Christmas	Indian/Cocos
Indian/Comoro	Indian/Kerguelen	Indian/Mahe	Indian/Maldives
Indian/Mauritius	Indian/Mayotte	Indian/Reunion	
Supported Time Zones - J, K, L			
Jamaica	Japan	Kwajalein	Libya
Supported Time Zones - M			
MET	MST	MST7MDT	
Supported Time Zones - Mexico			
Mexico/BajaNorte	Mexico/BajaSur	Mexico/General	
Supported Time Zones - N			
NZ	NZ-CHAT	Navajo	
Supported Time Zones - P			
PRC	PST8PDT	Poland	Portugal
Supported Time Zones - Pacific			
Pacific/Apia	Pacific/Auckland	Pacific/Chatham	Pacific/Chuuk
Pacific/Easter	Pacific/Efate	Pacific/Enderbury	Pacific/Fakaofo
Pacific/Fiji	Pacific/Funafuti	Pacific/Galapagos	Pacific/Gambier
Pacific/Guadalcanal	Pacific/Guam	Pacific/Honolulu	Pacific/Johnston
Pacific/Kiritimati	Pacific/Kosrae	Pacific/Kwajalein	Pacific/Majuro
Pacific/Marquesas	Pacific/Midway	Pacific/Nauru	Pacific/Niue
Pacific/Norfolk	Pacific/Noumea	Pacific/Pago_Pago	Pacific/Palau
Pacific/Pitcairn	Pacific/Pohnpei	Pacific/Ponape	Pacific/Port_Moresby
Pacific/Rarotonga	Pacific/Saipan	Pacific/Samoa	Pacific/Tahiti
Pacific/Tarawa	Pacific/Tongatapu	Pacific/Truk	Pacific/Wake
Pacific/Wallis	Pacific/Yap		
Supported Time Zones - R, S, T			
ROC	ROK	Singapore	Turkey
Supported Time Zones - U			
UCT	UTC	Universal	
Supported Time Zones - US			
US/Alaska	US/Aleutian	US/Arizona	US/Central
US/East-Indiana	US/Eastern	US/Hawaii	US/Indiana-Starke
US/Michigan	US/Mountain	US/Pacific	US/Pacific-New
US/Samoa			
Supported Time Zones - W, Z			
W-SU	WET	Zulu	

Appendix A. Best Practices

A.1 Multiple Queries

When there are multiple tasks to be performed they can be wrapped into a single request. For example, to get a list of files in Python, look up the device info, and look up system settings there would need to be three different requests:

```
<sci_request version="1.0">
  <send_message cache="false">
    <!-- list targets for query -->
    <targets>
      <device id="00000000-00000000-00000000-00000000" />
    </targets>
  </sci_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>

```

A.2 Reusing HTTP Session

A web service request when made must send the credentials via HTTP basic authentication. Subsequent calls however may use the session created. The benefit of doing this is performance, the authentication costs are negated. This does however add complexity to the code as HTTP cookie management must come into play.

An application must check the device stats state repeatedly in 30 second intervals looking for high CPU utilization of a device. The implementation in java can be implemented as follows:

```

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import org.apache.commons.codec.binary.Base64;

/**
 * DeviceUsagePoll can be used to poll the CPU utilization of a device. Uses a
 * naive approach to a cookie store which will save a Device Cloud session data to
 * avoid authentication overhead.
 *
 * Written for use with a Digi ConnectPort X2/4/8 product connected to
 * login.etherios.com.
 */
public class DeviceUsagePoll {
    /**
     * Authentication to be used on initial request
     */
    private String userPassword;
    /**
     * Device target for SCI requests
     */
    private String deviceId;
    /**
     * Flag for identifying if there has been a previous HTTP request to collect
     * session data
     */
    private boolean firstRequest;
    /**
     * Store for session identifiers
     */
    private String cookie;

    public DeviceUsagePoll(String userPassword, String deviceId) {
        super();
        this.userPassword = userPassword;
        this.deviceId = deviceId;
        this.firstRequest = true;
        this.cookie = "";
    }

    private String getDeviceUtilization() throws IOException {
        // Create url to Device Cloud server for a given web service request
        URL url = new URL("http://login.etherios.com/ws/sci");
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setDoOutput(true);
        conn.setRequestMethod("POST");
        // either authenticate for first request or use saved session
        if (firstRequest) {
            // replace with your username/password
            // can change this to use a different base64 encoder
            byte[] authRaw = Base64.encodeBase64(userPassword.getBytes());
            String enAuth = new String(authRaw).trim();
            conn.setRequestProperty("Authorization", "Basic " + enAuth);
        } else {
            // use saved session
            conn.setRequestProperty("Cookie", cookie);
        }
        // Send data to server
        conn.setRequestProperty("Content-Type", "text/xml");
        OutputStream os = conn.getOutputStream();
        OutputStreamWriter out = new OutputStreamWriter(os);
        out.write("<sci_request version='1.0'>\n");
        out.write("  <send_message cache='false'>\n");
        out.write("    <targets>\n");
        out.write("      <device id='" + deviceId + "'>\n");
        out.write("    </targets>\n");
        out.write("    <rci_request version='1.1'>\n");
        out.write("      <query_state><device_stats></query_state>\n");
        out.write("    </rci_request>\n");
        out.write("  </send_message>\n");
        out.write("</sci_request>\n");
        out.close();
        // Get input stream from response and convert to String
        // conn.disconnect();
        conn.setDoInput(true);
        if (conn.getResponseCode() == 401) {
            String msg = "HTTP response 401- Wrong credentials";
            throw new IOException();
        } else if (conn.getResponseCode() == 400) {
            throw new IOException("HTTP response 400 -Invalid device id?");
        }
    }
}

```

```

    }
    InputStream is = conn.getInputStream();
    Scanner isScanner = new Scanner(is);
    StringBuffer buf = new StringBuffer();
    while (isScanner.hasNextLine()) {
        buf.append(isScanner.nextLine() + "\n");
    }
    String responseContent = buf.toString();
    // extract CPU usage from response
    Pattern regex = Pattern.compile(".*<cpu>(.*?)</cpu>.*");
    Matcher match = regex.matcher(responseContent);
    match.find();
    // save cpu usage to be returned later
    String cpu = match.group(1);
    // store the session data if first request
    if (firstRequest) {
        for (int i = 0; i++) {
            String headerName = conn.getHeaderFieldKey(i);
            String headerValue = conn.getHeaderField(i);
            if (headerName == null && headerValue == null) {
                // No more headers
                break;
            }
            if ("Set-Cookie".equalsIgnoreCase(headerName)) {
                // Parse cookie
                String[] fields = headerValue.split(";\\s*");
                // fields[0] is the cookie value, like "SID=cc1"
                // split into name/value
                String[] cookieValue = fields[0].split("=");
                String name = cookieValue[0];
                String value = cookieValue[1];
                cookie += name + "=" + value + ";";
            }
        }
        this.firstRequest = false;
    }
    return cpu;
}

/**
 * Run the web service request
 */
public static void main(String[] args) {
    // =====Update with your information===== */
    String usernamePassword = "username:password";
    String deviceId = "00000000-00000000-00000000-00000000";
    /* ===== */
    // initialize poller
    DeviceUsagePoll poller = new DeviceUsagePoll(usernamePassword, deviceId);
    try {
        // poll every 10 seconds printing the result to standard out
        while (true) {
            String currentCpu = poller.getDeviceUtilization();
            System.out.println(currentCpu + "%");
            Thread.sleep(30000);
        }
        // catch lazy exception for example
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}

```

Appendix B. UI Descriptor Reference

B.1 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="">
      <automenue page="" dataRootDefault="settings" data="settings:*" readonly="false" />
    </menu>
  </navigation>
</ui>

```

B.1.1 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>

```

Will render 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 Device Cloud 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 Device Cloud 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).

B.1.2 Automenu

The automenu will render all settings for a device that has not been reserved by a different menu item via the data attribute.

Example:

```
<ui>
  <navigation>
    <menu id="advanced_cfg" name="Advanced Configuration">
      <automenu data="settings:*" readonly="false" />
    </menu>
  </navigation>
</ui>
```



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.

B.1.3 Page Templates

HTML templates.

Example:

```
<ui>
  <content>
    <page id="test_page" help="test_page_help">
      <b>My IP setting:</b>
      <property rcId="settings:mgmtconnection/1/serverAddress" />
      <hr />
      <h1>Advanced:</h1>
      <unprocessed>
        <exclude rcId="settings:mgmtconnection/1/timedConnectionPeriod" />
      </unprocessed>
    </page>
  </content>
</ui>
```

B.1.3.1 Attributes

There are two attributes you can define for the page element in the content:

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.

B.1.3.2 Page Contents

The page contains `<html>` 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 Device Cloud Manager server:</b>
  <property rcId="settings:mgmtconnection/1/serverAddress" />
  <hr />
  <h1>Advanced:</h1>
  <unprocessed>
    <exclude rcId="settings:mgmtconnection/1/timedConnectionPeriod" />
  </unprocessed>
</page>
```

Property tag

The property tag is a special element that will be replaced with a UI field for a setting given by the `rcId` attribute. The type (text box, drop down, etc.) of the field would be determined by the RCI descriptor for the setting. If no descriptor is available it will be a text box.

```
<page id="test_page">
  <b>My Device Cloud Manager server:</b>
  <property rcId="settings:mgmtconnection/1/serverAddress" />
</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 setting from this list.

```
<page id="test_page">
  <h1>Advanced:</h1>
  <unprocessed>
    <exclude rcId="settings:mgmtconnection/1/timedConnectionPeriod" />
  </unprocessed>
</page>
```

B.1.4 Help Templates

HTML templates

Example:

```
<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 `<html>` that will be displayed in popup when the help is clicked in the properties page.

Appendix C. Transport Protocols for the Monitor API

The Web Services Monitor API supports two transport options for pushing published data to the application - TCP and HTTP. The following sections describe the protocol details of each of these transport protocols.

C.1 TCP Transport Protocol

This section highlights the details associated with a standard TCP/IP or SSL socket connection between the client application and Device Cloud. Since authentication messages will be flowing across the socket, SSL is strongly recommended. Standard TCP/IP connection is available but should only be used for debugging and troubleshooting.

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 Device Cloud's server. SSL monitor sockets should be made to port 3201 while unsecure TCP sockets should be made to port 3200.

Once the socket connection has been made, the customer application must send a `ConnectRequest` message through it to Device Cloud's server. The server will authenticate the request and send back a response. If the connect request succeeds, the server will begin sending `PublishMessages` to the customer application as events matching the monitor configuration occur. Upon receiving `PublishMessages`, the customer application should acknowledge receipt of those messages to the server with a `PublishMessageReceive` message.

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. The customer application can reactivate the monitor socket in the same manner as the initial connection.

C.1.1 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.

C.1.1.1 Framing

All messages between the client application and Device Cloud's 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

C.1.2 Messages

C.1.2.1 ConnectRequest

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

Header [6 Bytes] Type=0x0001
Payload:

- Example:**

[illegible]

```
Type: 0x0001
Size: 0x00000013
ProtocolVersion: 0x0001
UsernameSize: 0x0005
Username: 0x7065f737369 (pepsi)
PasswordSize: 0x0004
Password: 0x636f6c61 (cola)
MessageId: 0x00000104
```

The response to `ConnectRequest`, sent from Device Cloud 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 Device Cloud is speaking.

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

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

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

As monitored events occur, Device Cloud will send `PublishMessage` messages to the client application.

- 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)

offset	bytes															
0x0000	type		size					block id		count		cmp	fmt	payload size		
	00	03	00	00	02	15	01	A7	00	02	00	00	00	00	02	05
0x0010	payload data															

- Monitor-Aggregate-Count the number of published 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. Refer to the [Monitor Published Event Payload](#) section for further details on the payload data format.

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

C.3 Monitor Published Event Payload

C.3.1 Payload Data

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",
        "...": "..."
      }
    }
  }
}
```

Appendix D. Simple HTTP Device Interface

Devices implement [Ethereos Cloud Connector](#) to integrate with Device Cloud. Cloud Connector enables devices to fully integrate with Device Cloud offering many rich features. More information is available online at www.etheros.com/devicecloud/connector.

Cloud Connectors are available in several languages and operating environments such as C and Java for Android, and must be ported to target platforms. There are times when it is not practical to port the connector to a platform (such as because it is not based on C or Java). Sometimes a user needs to only upload data and does not need to utilize all of the features of Cloud Connector. For these users the Simple HTTP Device Interface should be used.

D.1 Introduction

The Simple HTTP Device Interface can be used by any device, program, or anything that can perform simple HTTP operations. The Simple HTTP Device Interface allows a client to upload data to Device Cloud and offers a simple way for devices to retrieve messages from the server. The Simple HTTP Device uses only basic HTTP operations to perform all of its interactions with Device Cloud.

D.2 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 Device Cloud by performing a POST to `/ws/DeviceCore` and specifying the `<provisionId>` and `<dpCurrentConnectPw>` values. See the [DeviceCore](#) section, specifically Example #4 for more information.

D.2.1 Uploading Data to Device Cloud

PUT `/ws/Messaging/<path>[?append=true]`

Body: Device Cloud message contents

Header:

- Content-Type: (optional)
- delete: (optional) See Delete below.
- deleted: (optional) See Delete below.

Parameters:

- append=true (optional): Appends the message to an existing file if it exists or creates it if it does not exist. If this option is omitted, the file is replaced if it exists.

A client sends Device Cloud a message in the form of a file located at `<path>`. `<path>` can be a simple file name, or a directory path. Device Cloud places the file in the specified path within the device's Data Service which can be accessed from Device 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).

D.2.2 Sending a Message to a Device

The Simple HTTP Device Interface supports a simple mechanism that allows users to retrieve messages from Device Cloud.

A folder for the device called "inbox" is created in Device Cloud's 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]

Header:

- delete: (optional) See [Deleting a Message from a Device Inbox](#) below.
- deleted: (optional) See [Deleting a Message from a Device Inbox](#) below.

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)...

...

A GET to /ws/Messaging/message1 returns the contents of message1.

D.2.3 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.

Example:

The following example shows the steps for creating a Device ID that is used in a simple python program to upload a data file to Device Cloud.

To create a Device ID (if you haven't already):

1. Sign in to your Device Cloud user account
2. Open the **My Account** page by clicking the Account Settings menu within the Admin tab.
3. Within the Vendor Information section of the page, copy your Vendor ID number if one exists. If one does not exist, click the button to create one.
 - a. For this example, we will use the Vendor ID: 0x0100001D. You will need to replace this with number with your own Vendor ID.
 - b. A Vendor ID can also be created or retrieved using the /ws/DeviceVendor web service. See the [DeviceVendor](#) section for more information.
4. Open the **API Explorer** by clicking the API Explorer menu within the Documentation tab.
5. To have Device Cloud generate a Device ID for you, perform a POST on /ws/DeviceCore using the following specifics:
 - a. Path: /ws/DeviceCore
 - b. HTTP Method: POST
 - c. Paste the following into the text area:

```
<DeviceCore>
  <provisionId>0x0100001D</provisionId>
  <dpCurrentConnectPw>DevicePassword123</dpCurrentConnectPw>
  <dpRestrictedStatus>DevicePassword123</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 Device Cloud
 - <grpPath/>: instructs Device Cloud 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. Click **Send** within the API Explorer toolbar to execute the operation.
 7. The server will respond with a 201 (displayed within the Web Services Responses window); 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 once again.
 11. Open the result and find the devConnectwareID; this is your newly created Device ID.

Now, copy the python program below into an editor, filling in the Device ID and device password used in the steps above, then run the program.

The program uploads 10 sample readings. The results can be viewed in Device 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 Device Cloud 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 = "login.etherios.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)

```

Appendix E. ISO 8601 Date & 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. Currently, Date and Duration attributes can be found within the [Scheduling Attributes](#) and [Data Points](#) sections. The information within this Appendix is meant to help you gain a better understanding of ISO 8601 format in order to properly express Date and Duration attributes in ISO 8601 format.

E.1 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, Device Cloud only supports the following format:

Complete date plus hours, minutes and seconds:
 YYYY-MM-DDThh:mm:ssTZD (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)

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

E.2 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.