



iDigi[®] Web Services Programming Guide

90002008_B

©2011 Digi International Inc.
All Rights Reserved.

Digi, Digi International, the Digi logo, the Digi website, iDigi, the iDigi logo, iDigi Dia, iDigi Manager Pro, iDigi Web Services API, XBee, and ConnectPort X are trademarks or registered trademarks of Digi International, Inc. in the United States and other countries world wide.

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.



Table of Contents

Chapter 1: Introduction	5
1.1 Overview	5
1.2 Applications of iDigi	5
1.3 Web Service Categories	6
Chapter 2: Concepts	7
2.1 Subscriptions	7
2.2 Device IDs	7
2.2.1 Device ID Assignment Convention	8
2.2.1.1 48 bit MAC Address Format	8
2.2.1.2 GSM IMEI	8
2.2.1.3 CDMA ESN/MEID	8
2.2.1.4 Orbcomm Addresses	9
2.3 Embedded Device Development	9
2.4 Data File Caching (Storage)	10
2.4.1 Seamless Data Transfer from Device to iDigi	10
2.4.2 Path Information	10
2.5 Device Information Caching	11
2.5.1 Device Meta Data Cache	11
2.5.2 Device Data Cache	13
2.5.3 XBee Node Cache	13
2.5.4 XBee Data Cache	13
2.5.5 Delta Cache Mechanism	14
Chapter 3: Writing Web Services Client Applications	15
3.1 In a Web Browser	15
3.2 In the iDigi Web Application	15
3.3 Python	16
3.4 Java	17



Chapter 4: Resource Web Services	19
4.1 URL Specification	19
4.2 Resource Web Service CRUD Conventions	19
4.2.1 Resource Overview	20
Chapter 5: SCI (Server Command Interface)	26
5.1 Introduction	26
5.2 Synchronous Request	27
5.3 Asynchronous Request	28
5.3.1 Performing an Asynchronous Request	28
5.3.2 Retrieve Status	28
5.3.2.1 Status for a Particular Job	28
5.3.2.2 Overall Status of Outstanding Jobs	29
5.3.3 Cancel a Request or Delete the Results	30
5.4 Anatomy of an SCI Request	30
5.5 Available Operations	30
Chapter 6: Data Files (Storage)	35
6.1 Introduction	35
Chapter 7: Security	39
7.1 Initial Password	39
7.2 Changing a Password	39
Chapter 8: Core API Technical Reference	41
8.1 DeviceCore	41
8.2 DeviceInterface	42
8.3 DeviceMetaData	44
8.4 DeviceVendor	45
8.5 DeviceVendorSummary	45
8.6 FileData	46
8.7 NetworkInterface	50
8.8 XbeeCore	50
Chapter 9: Energy API Technical Reference	53
9.1 Manufacturer Specific Attributes	53

9.2 Energy APIs	53
9.3 XbeeAttributeCore	54
9.4 XbeeAttributeFull	56
9.5 XbeeAttributeDataCore	56
9.6 XbeeAttributeDataFull.....	59
9.7 XbeeAttributeDataHistoryCore	59
9.8 XbeeAttributeDataHistoryFull	61
9.9 XbeeAttributeReportingCore	61
9.10 XbeeEventDataCore	63
9.11 XbeeEventDataFull	66
9.12 XbeeEventDataHistoryCore	66
9.13 XbeeEventDataHistoryFull	66
Chapter 10: Dia Web Services API.....	67
10.1 Dia Web Services API	67
Chapter 11: iDigi SMS.....	72
11.1 Receiving iDigi SMS Messages	72
11.2 Sending iDigi SMS Messages via Web Services.....	72
11.3 iDigi SMS Command Children.....	73
11.4 Shoulder-Tap iDigi SMS Support	74
11.4.1 Configure iDigi with the Phone Number of the iDigi Device	74
11.4.2 Configure Device to Receive iDigi SMS Commands	78
11.4.3 RCI for iDigi SMS	79
11.4.4 Send iDigi SMS Request Connect	80
11.4.5 Wait for Device to Connect	81
11.4.6 Send a Disconnect	82
Appendix A: Best Practices.....	83
A.1 Multiple Queries	83
A.2 Reusing HTTP Session.....	83
Appendix B: UI Descriptor Reference	87
B.1 Menu Templates.....	87
B.1.1 Menu Element	88

B.1.2 Automenu	90
B.1.3 Page Templates	91
B.1.3.1 Attributes	91
B.1.3.2 Page Contents	91
B.1.4 Help Templates	92



1. Introduction

1.1 Overview

The iDigi[®] platform is a machine-to-machine cloud-based network operating platform that includes a variety of application programming Interfaces (API's). iDigi 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 customer applications. An important part of this communication is the transfer of data from a device to the server. Users can write custom code that run on devices to pass data (messages) as well as send data to a temporary data cache on the iDigi platform to be available for retrieval by web services clients. Digi makes this easy by providing development kits, smart energy devices, and the iDigi[®] Dia (Device Integration Application).

1.2 Applications of iDigi

There are many real world applications where iDigi can be very beneficial. One such application is in energy management where power utilities can provide remote access and management to appliances in homes or businesses that put a heavy load on the energy grid. Other applications are fleet management where a fleet of vehicles can be monitored from a remote location and building automation where lighting can be controlled to save energy. Also, the monitoring and data collection of a sensor network can be made easier with iDigi. In addition to handling data from your devices, iDigi supports device management such as upgrading firmware and making configuration changes.

iDigi 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.



1.3 Web Service Categories

This document describes iDigi Web Services. Web services allow users to write custom applications for remote management, monitoring, data acquisition and other purposes. There are several categories of web services provided by iDigi:

Resource based web services represent data remotely managed by iDigi.

SCI is a server command interface web service to request jobs either synchronously or asynchronously and can send commands to the managed resources connected to iDigi.

Data 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.

Security web service allows an application to update the EDP protocol password of a device.

2. Concepts

2.1 Subscriptions

Subscriptions in iDigi control what features are available on a customer basis. In `developer.idigi.com`, users are automatically subscribed to all available iDigi services. In production environments, subscriptions are used to turn on and off iDigi services.

Access to web services are 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 iDigi. 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. The following dash should be omitted as well. At least one group must be specified.

Example Device ID Abbreviations

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
00000000-00000000-00000000-00000000	00000000-00000000-00000000 00000000-00000000 00000000

2.2.1 Device ID Assignment Convention

The Device ID is generated by a seeding interface 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 Orbcom global unique id

2.2.1.1 48 bit MAC Address Format

- 12 hex digits
- mapping to CWID
- top 64bits set to zero
- lower 64 bits using EUI-64 format (MAC-48 extension)

Example MAC: 112233:445566

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

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.

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, since IMEI and MEID will have separate specifications for the Device ID.



Example ESN-Hex: MM-SSSSSS

Device Id mapping: 00020000-00000000-00000000-MMSSSSSS

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 DeviceID 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 contain firmware that is iDigi enabled. Third party device developers can create devices that are iDigi enabled using development kits.

When a device connects to iDigi, 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.

The following process is used to create an iDigi enabled device.

1. Install a development kit.
2. Create an account on developer.idigi.com.
3. On the My Account page, generate a Vendor ID.
4. Put the Vendor ID in the application for the device and configure the device type to something meaningful and specific for the device.
5. Build the application and deploy it to the device.
6. When the application starts it will connect to iDigi. iDigi sees it is a new device type for that Vendor ID and queries the device for its descriptors.
7. Navigate to the Device page in the iDigi web application and see the device show up in the device list. View the device properties to see the descriptors render the settings/state of the device.
8. On the iDigi Device page select the Edit UI descriptors menu. Create a view descriptor. Re-open the device properties page and see the settings/state rendered with additional format provided by the view descriptor.
9. Edit the application to have custom settings/state exposed. Build and deploy the application to the device.
10. On the iDigi Device page, select the Refresh descriptors menu to retrieve the latest descriptors.
11. Navigate to the device properties page and see custom settings/state exposed in the UI.

2.4 Data File Caching (Storage)

iDigi's data service facilitates data collection from remote devices. Devices can push data files up to the server that are cached temporarily in a database on the iDigi platform. The 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, like displaying it in a graph on a web page. Files and collections can also be accessed through the user portal view of the iDigi platform as well as via Web Services.

2.4.1 Seamless Data Transfer from Device to iDigi

A device can easily send any data to the iDigi platform with a simple function call. The device and iDigi take care of the details. With this feature, a user can decide which data to send to the server and how often to send it. The data that is sent by the device will usually be XML data and it will be stored in a database on the server.

2.4.2 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, iDigi caches a variety of device related data. Some of this data is related to a specific device and some of it is meta data about entire classes of devices. Some data is related to remote nodes located behind a gateway device. The server has numerous caching mechanisms to organize and store this data. The sections below describe these mechanisms and the data they store.

Before digging into each cache in detail, the general approach used by the iDigi server is to accept a request for information and attempt to satisfy that request from one or more of the servers data caches. If the request cannot be satisfied from the cache, then the request may be sent on to one or more of the connected devices to get the requested information. When the results are returned, the server updates its caches to include the new information so that future requests may be satisfied more quickly. Finally, the results are returned to the caller - who generally isn't aware if the results came from the cache or if they came directly from the device.

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

Finally, 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, the iDigi 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) identify 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 in fact, 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 radios ExtendedAddress 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.

2.5.5 Delta Cache Mechanism

Storing the complete settings and state of every device connected to the system is convenient for many operations however it becomes very expensive in terms of the amount of data stored in the server. Additionally this is expensive in terms of the amount of redundant data being stored, since very few device configuration options are ever changed from their default setting. We reduce this expense by storing the default settings for every type of device and then simply storing the delta (i.e. changed) settings for each device instance. This requires our cache processing to compare and/or merge the default settings with the device instance settings as the cache requests and/or updates are processed.

When fetching settings of a device, the cache manager locates the delta and default settings for the specified device, and merges them to present the total configuration settings to the user. When updating settings of a device, the update is sent to the device and upon successful application by the device, the server updates the delta cache by comparing the updated settings to the defaults and storing the difference in the delta.

When using the SCI interface, the user is always interacting with the combined default/delta view.

3. Writing Web Services Client Applications

The iDigi server provides a REST-style API over HTTP (or HTTPS). Users can write HTTP clients in their preferred programming language that get data from the iDigi platform 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 the iDigi platform 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 the iDigi Web Application

The iDigi web application has a web services console 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.

The iDigi web application has a web services console 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 httpplib
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 = httpplib.HTTP("developer.idigi.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.

The iDigi web application has a web services console 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 an iDigi web service.
 */
public class WebServiceRequest {

    /**
     * Run the web service request
     */
    public static void main(String[] args) {
        try {
            // Create url to the iDigi server for a given web service request
            URL url = new URL("http://developer.idigi.com/ws/sci");
            HttpURLConnection conn = (HttpURLConnection)url.openConnection();

            conn.setDoOutput(true);
            conn.setRequestMethod("POST");
            // replace with your username/password
            String userpassword = "YourUsername:YourPassword";
            // can change this to use a different base64 encoder
            String encodedAuthorization = Base64.encodeBase64String(
                userpassword.getBytes()
            ).trim();

            conn.setRequestProperty("Authorization", "Basic "+ encodedAuthorization);
            // Send data to server
            conn.setRequestProperty("Content-Type", "text/xml");
            OutputStreamWriter out = new OutputStreamWriter(conn.getOutputStream());
            out.write("<sci_request version=\"1.0\"> \r\n");
            out.write("    <send_message> \r\n");
            out.write("        <targets> \r\n");
            out.write("            <device id=\"00000000-00000000-00000000-00000000\"/> \r\n");
            out.write("        </targets> \r\n");
            out.write("        <rci_request version=\"1.1\"> \r\n");
            out.write("            <query_state/> \r\n");
```

```
out.write("    </rci_request>\r\n");
out.write(" </send_message>\r\n");
out.write("</sci_request>\r\n");
out.close();
// Get input stream from response and convert to String
conn.disconnect();
conn.setDoInput(true);
InputStream is = conn.getInputStream();
Scanner isScanner = new Scanner(is);
StringBuffer buf = new StringBuffer();
while(isScanner.hasNextLine()) {
    buf.append(isScanner.nextLine() + "\n");
}
String responseContent = buf.toString();
// Output response to standard out
System.out.println(responseContent);
} catch (IOException e) {
    // Print any exceptions that occur
    e.printStackTrace();
}
}
```

4. Resource Web Services

4.1 URL Specification

iDigi web services are RESTful in nature. Every URL relates to a specific resource or list of resources.

§ Example: DeviceCore URLs

The following URL retrieves a list of devices,

<http://developer.idigi.com/ws/DeviceCore>

whereas this next URL retrieves a single device with id 1.

<http://developer.idigi.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 Services. 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	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
NetworkInterface	X	X	X	X	DM	Device modem list
XbeeCore	X				DM	XBee nodes and properties

POST

HTTP POST is used to add resources to your account.

POST URI 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 userid/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 {Base 64 encoded password} Effect: Authorizes resource access

Return codes:

- 200 (OK)
- 400 (Bad request) The request is invalid.
- 401 (Unauthorized) The userid/password is invalid.
- 403 (Forbidden) Access to the resource is not authorized. You may need a subscription.
- 500 (Internal server error) The request can not 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 the iDigi database)

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="ISO-8859-1"?>
<result>
  <resultTotalRows>13</resultTotalRows>
  <requestedStartRow>11</requestedStartRow>
  <resultSize>2</resultSize>
  <requestedSize>2</requestedSize>
  <remainingSize>0</remainingSize>
  <DeviceCore>
    <id>
      <devId>155</devId>
      <devVersion>0</devVersion>
    </id>
    <devRecordStartDate>2010-06-25T21:28:00Z</devRecordStartDate>
    <devMac>00:40:9D:3D:71:A6</devMac>
    <devConnectwareId>00000000-00000000-00409DFF-FF3D71A6</devConnectwareId>
    <cstId>3</cstId>
    <grpId>3</grpId>
    <devEffectiveStartDate>2010-06-25T21:28:00Z</devEffectiveStartDate>
    <devTerminated>false</devTerminated>
    <dvVendorId>0</dvVendorId>
    <dpDeviceType>ConnectPort X2</dpDeviceType>
    <dpFirmwareLevel>34209795</dpFirmwareLevel>
    <dpFirmwareLevelDesc>2.10.0.3</dpFirmwareLevelDesc>
    <dpRestrictedStatus>0</dpRestrictedStatus>
    <dpLastKnownIp>10.20.1.161</dpLastKnownIp>
    <dpGlobalIp>10.20.1.161</dpGlobalIp>
    <dpConnectionStatus>1</dpConnectionStatus>

```



```
<dpLastConnectTime>2010-06-28T13:35:00Z</dpLastConnectTime>
<dpContact/>
<dpDescription>EMS - Aux gateway #2 - Test certificate</dpDescription>
<dpLocation>Jeff's office</dpLocation>
<dpPanId>0x134f</dpPanId>
<xpExtAddr>00:13:a2:00:40:5c:0a:ba</xpExtAddr>
<dpServerId>ClientID[3]</dpServerId>
</DeviceCore>
<DeviceCore>
  <id>
    <devId>156</devId>
    <devVersion>0</devVersion>
  </id>
  <devRecordStartDate>2010-06-25T20:46:00Z</devRecordStartDate>
  <devMac>00:40:9D:29:5B:4C</devMac>
  <devConnectwareId>00000000-00000000-00409DFF-FF295B4C</devConnectwareId>
  <cstId>3</cstId>
  <grpId>3</grpId>
  <devEffectiveStartDate>2010-06-25T20:46:00Z</devEffectiveStartDate>
  <devTerminated>false</devTerminated>
  <dvVendorId>0</dvVendorId>
  <dpDeviceType>Digi Connect WAN VPN</dpDeviceType>
  <dpFirmwareLevel>34014219</dpFirmwareLevel>
  <dpFirmwareLevelDesc>2.7.2.11</dpFirmwareLevelDesc>
  <dpRestrictedStatus>0</dpRestrictedStatus>
  <dpLastKnownIp>10.20.1.144</dpLastKnownIp>
  <dpGlobalIp>10.20.1.144</dpGlobalIp>
  <dpConnectionStatus>1</dpConnectionStatus>
  <dpLastConnectTime>2010-06-28T13:35:00Z</dpLastConnectTime>
  <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 for 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 userid/password is invalid.
- 403 (Forbidden) Access to the resource is not authorized. You may need a subscription.
- 500 (Internal server error) The request can not 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 userid/password is invalid.
- 403 (Forbidden) Access to the resource is not authorized. You may need a subscription.
- 500 (Internal server error) The request can not be handled due to a server error. Wait and try again.

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

5. SCI (Server Command Interface)

5.1 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`.



5.2 Synchronous Request

POST the following to: <http://developer.idigi.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>
</sci_reply>
```

5.3 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
```

5.3.1 Performing an Asynchronous Request

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

```
<send_message synchronous="false">
```

the response then has the form:

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

where **job_id** identifies the request you submitted.

5.3.2 Retrieve Status

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

5.3.2.1 Status for a Particular Job

Do an HTTP GET on http://developer.idigi.com/ws/sci/{job_id}

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

5.3.2.2 Overall Status of Outstanding Jobs

Do an HTTP GET on <http://developer.idigi.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>
    <csId>0</csId>
    <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

5.3.3 Cancel a Request or Delete the Results

Do an HTTP DELETE on `http://developer.idigi.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 stop starting.

5.4 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}" />` or `<device id="all" />`

payload is dependent on the operation

5.5 Available Operations

Four 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

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">
```

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 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. By default this value changes based on the type of SCI request, number of targets etc.

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

- *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.

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. iDigi 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>
```



update_firmware

This is used to update the firmware of one or more devices. The firmware image needs to be base 64 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 the iDigi platform completed successfully.

It is still possible for the process to fail between the iDigi platform 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 looks like:

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

and the reply looks like:

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

disconnect

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

A request follows this format:

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

and a response looks like:

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

query_firmware_targets

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

A request follows this format:

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

and a response looks like:

```
<sci_reply version="1.0">
  <query_firmware_targets>
    <device id="00000000-00000000-00000000-00000000">
      <targets>
        <target number="0">
          <name>Firmware Image</name>
          <pattern>image\.bin</pattern>
          <version>7.5.0.11</version>
          <code_size>2162688</code_size>
        </target>
      </targets>
    </device>
  </query_firmware_targets>
</sci_reply>
```

```
</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. Data Files (Storage)

6.1 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	Get 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 section Uploading files with POST below.

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 can not 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 can not 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}

Example:

/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



7. Security

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

Setting an EDP password for a device will cause iDigi to verify the identity of the device on connect. If a device has a password set in iDigi, it must be configured to send up that password, or iDigi will not allow the connection. By default, iDigi 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.

7.1 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.

7.2 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 can not 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.

8. Core API Technical Reference

8.1 DeviceCore

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

HTTP operations supported:

- GET : Display a list of devices provisioned in your account.
- POST : Add a device to your account.
- 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
- cstId - the automatically generated ID assigned to a customer
- grpId - the automatically generated ID assigned to a customer's group
- devEffectiveStartDate - the date the device was provisioned in iDigi
- 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 iDigi 0 is OK, 1 is not allowed to connect
- 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 iDigi 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



- 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

Examples:

See the POST and GET sections above for examples of using DeviceCore.

8.2 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 DeviceCore and NetworkInterface tables 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.

Examples:

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="ISO-8859-1"?>
<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>
    <cstId>7</cstId>
    <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>
  <csiId>7</csiId>
  <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>
  <csiId>7</csiId>
  <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>
<cstId>7</cstId>
<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>
```

8.3 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

8.4 DeviceVendor

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.

Elements include:

- dvVendorId - integer representation of this vendor ID
- dvVendorIdDesc - hex representation of this vendor ID
- cstId - iDigi id of customer owning this vendor ID
- dvDescription - Information describing this vendor ID
- dvRegistrationDate - When the vendor ID was registered

8.5 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 - iDigi id of customer owning this vendor ID
- dvDescription - Information describing this vendor ID
- dmUiDescriptorCount - Number of view descriptors that exist for this device type

8.6 FileData

iDigi 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. The files will generally be available for 24 days. After that time, the data will be pruned from the database.

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
- cstId - Indicates the iDigi id of customer owning the file
- fdCreatedDate - Indicates the date that the file was first uploaded to iDigi
- fdLastModifiedDate - Indicates the date that the file was last modified
- fdContentType - Indicates the type of data stored in the file
- fdSize - Indicates the size of the file contents - in bytes
- fdType - 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="ISO-8859-1"?>
<result>
  <resultTotalRows>455747</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>1000</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>454747</remainingSize>
  <FileData>
    <id>
      <fdPath>/db/SB723050334974_Digi_International/00000000-00000000-
00409DFF-FF640005</fdPath>
```

```

        <fdName>RPC_response-1297463631.0-0001-
        received_attribute_report.xml</fdName>
    </id>
    <cstId>3439</cstId>
    <fdCreatedDate>2011-02-11T22:34:25Z</fdCreatedDate>
    <fdLastModifiedDate>2011-02-11T22:34:25Z</fdLastModifiedDate>
    <fdContentType>application/xml</fdContentType>
    <fdSize>506</fdSize>
    <fdType>file</fdType>
</FileData>
...
<FileData>
    <id>
        <fdPath>/db/SB723050334974_Digi_International/00000000-00000000-
        00409DFF-FF640005</fdPath>
        <fdName>RPC_response-1297463631.0-0003-
        received_attribute_report.xml</fdName>
    </id>
    <cstId>3439</cstId>
    <fdCreatedDate>2011-02-11T22:34:25Z</fdCreatedDate>
    <fdLastModifiedDate>2011-02-11T22:34:25Z</fdLastModifiedDate>
    <fdContentType>application/xml</fdContentType>
    <fdSize>506</fdSize>
    <fdType>file</fdType>
</FileData>
</result>

```

Example #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 iDigi after the specified date.

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

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

Example #3

By specifying the option embed="true" the content of the files will be embedded in the meta-data in base64 format. For example, to get all files and their content after a specified date:

```
GET /ws/FileData?condition=fdName like 'sample%' and dType='file' and fdLastModified-
Date>'2010-11-24T22:25:04Z'&embed=true
```

Returns:

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<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 #4

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 #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:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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>VGhpcyBpcyB0aGUgdGV4dCBvZiB5b3VyIHRlc3QgZmlsZSE=</fdData>
  </FileData>
</result>
```



8.7 NetworkInterface

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.

NetworkInterface has the following new attributes to allow users to provision iDigi devices with SMS capability:

- Phone number for SIM 1
- Carrier ID for SIM 1
- You can update NetworkInterface with this information via the web services interface.

8.8 XbeeCore

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

HTTP operations supported:

- GET : Display a list of ZigBee nodes in your account.

Elements include:

- cstId - iDigi id of customer owning the gateway discovering the node
- grpId - iDigi id of the customer group owning the gateway
- 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
- xpUpdateTime -time the node was last discovered

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'
```

which returns:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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>
    <cstId>3</cstId>
    <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>
  </XbeeCore>
  <XbeeCore>
    <xpExtAddr>00:13:A2:00:40:27:03:93</xpExtAddr>
    <devConnectwareId>00000000-00000000-00409DFF-FF702005</devConnectwareId>
    <cstId>3</cstId>
    <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>
  </XbeeCore>

```

```
<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>
<cstId>3</cstId>
  <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>
```

9. Energy API Technical Reference

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. Let's look at an 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

Energy APIs

Resource Path	Get	Post	Put	Delete	Description
XbeeAttributeCore	X				XBee Attribute Names
XbeeAttributeFull	X				XBee Attribute Names
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.
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

XbeeAttributeData is used by clients to read and update attribute values of a node/endpoint/cluster in a HAN network. This API can be used in conjunction with the XbeeAttributeReportingCore web service 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:

- cstId - iDigi 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 0xFFFE
- 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:

```
<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>
    <cstId>3</cstId>
    <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>
    <cstId>3</cstId>
    <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>
<cstId>3</cstId>
<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. This API can be used in conjunction with the XbeeAttributeReportingCore web service 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` - iDigi 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 0xFFFFE
- `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>
    </id>
  </XbeeAttributeDataCore>
</result>
```

```

        <xaAttributeId>0</xaAttributeId>
    </id>
    <cstId>3</cstId>
    <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>
    <cstId>3</cstId>
    <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>
    <cstId>3</cstId>
    <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>
  <cstId>3</cstId>
  <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

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:

- cstId - iDigi 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
- xadUpdateTime - time that the report configuration was last reported or set

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 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="ISO-8859-1"?>
<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>
    <cstId>3</cstId>
    <devConnectwareId>00000000-00000000-00000000-00000000</devConnectwareId>
    <xeProfileId>265</xeProfileId>
    <xeDeviceId>1282</xeDeviceId>
    <xeDeviceVersion>1</xeDeviceVersion>
    <xedCommandId>0</xedCommandId>
  </XbeeEventDataCore>
</result>
```

```

<xedPayload>
  <record type="PublishPriceRecord">
    <current_timetype="int">0x140EC5BF</current_time>
    <provider_idtype="int">0xFFFFFFFF</provider_id>
    <duration_in_minutestype="int">0x2D0</duration_in_minutes>
    <unit_of_measuretype="int">0x0</unit_of_measure>
    <price_trailing_digit_and_price_tiertype="int">
      0x33
    </price_trailing_digit_and_price_tier>
    <start_time type="int">0x0</start_time>
    <number_of_price_tiers_and_register_tiertype="int">
      0x43
    </number_of_price_tiers_and_register_tier>
    <alternate_cost_deliveredtype="int">0xFFFFFFFF</
    alternate_cost_delivered>
    <currency type="int">0x348</currency>
    <issuer_event_idtype="int">0x3344</issuer_event_id>
    <alternate_cost_unittype="int">0x1</alternate_cost_unit>
    <alternate_cost_trailing_digittype="int">
      0xFF
    </ alternate_cost_trailing_digit>
    <price_ratio type="int">0xFF</price_ratio>
    <generation_pricetype="int">0xFFFFFFFF</generation_price>
    <price type="int">0xA</price>
    <generation_price_ratiotype="int">0xFF</
    generation_price_ratio>
    <rate_labeltype="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>
  <cstId>3</cstId>
  <devConnectwareId>00000000-00000000-00000000-00000000</devConnectwareId>
  <xeProfileId>265</xeProfileId>
  <xeDeviceId>1282</xeDeviceId>
  <xeDeviceVersion>1</xeDeviceVersion>
  <xedCommandId>0</xedCommandId>
</xedPayload>
  <record type="LoadControlEventRecord">
    <duration_in_minutestype="int">0x258</duration_in_minutes>
    <cooling_temperature_offsettype="int">0xFF</
    cooling_temperature_offset>
    <event_controltype="int">0x3</event_control>
    <start_time type="int">0x0</start_time>
    <device_classtype="int">0xFF</device_class>
  </record>
</XbeeEventDataCore>

```

```

        <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_percent type="int">
            -0x80
        </average_load_adjustment_percent>
        <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>
    <cstId>3</cstId>
    <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_minutestype="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>

```

9.11 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

9.12 XbeeEventDataHistoryCore

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

HTTP operations supported:

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

9.13 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 Web Services API

10.1 Dia Web Services API

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

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

NOTE: For this release, only the idigi_db presentation data is stored in the tables. The RCI presentation data queried via iDigi 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, iDigi may not be able to parse the sample data uploaded from the device. If the idigi_db presentation cannot be parsed by iDigi, the sample data won't be stored in the Dia tables; instead it will default to being stored as a file.

Dia Web Service Details

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

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

NOTE: For this release, only the idigi_db presentation data is stored in the tables. The RCI presentation data queried via iDigi 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, iDigi may not be able to parse the sample data uploaded from the device. If the idigi_db presentation cannot be parsed by iDigi, the sample data won't be stored in the Dia tables; instead it will default to being stored as a file.

DiaChannelDataFull

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

HTTP operations supported:



"GET: Display a list of the most recent dia channel data values

"DELETE

DiaChannelDataHistoryFull

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

HTTP operations supported:

"GET: Display a list of the historical Dia channel data values

"DELETE

Elements include:

- cstId - iDigi id of customer owning the gateway
- devConnectwareId - Device ID of the gateway
- xpExtAddr - ZigBee 64bit extended address of node
- 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 time-stamp 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 iDigi for historical records onlyExamples

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>
  <xpExtAddr>00:13:A2:00:40:52:C7:7D</xpExtAddr>
  <dcdUpdateTime>2011-01-17T16:35:21Z</dcdUpdateTime>
  <dcdStringValue>248650</dcdStringValue>
  <dcdIntegerValue>248650</dcdIntegerValue>
</DiaChannelDataFull>
</result>
```

This represents the following Dia iDigi_db presentation data pushed from the device with id '00000000-00000000-00409DFF-FF32E1F7':

```
<idigi_data>
  <sample>
    <name>template.counter</name>
    <value>248650</value>
    <unit></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)

```
<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>
    <xpExtAddr>00:13:A2:00:40:52:C7:7D</xpExtAddr>
    <dcDataType>0</dcDataType>
    <dcUpdateTime>2011-02-03T09:53:57Z</dcUpdateTime>
    <dcStringValue>1200157</dcStringValue>
    <dcIntegerValue>1200157</dcIntegerValue>
  </DiaChannelDataHistoryFull>
  <DiaChannelDataHistoryFull>
    <id>
      <devConnectwareId>00000000-00000000-00409DFF-FF32E1F7</devConnectwareId>
      <ddInstanceName>template</ddInstanceName>
      <dcChannelName>counter</dcChannelName>
      <dcdhId>20506</dcdhId>
    </id>
    <cstId>12</cstId>
    <xpExtAddr>00:13:A2:00:40:52:C7:7D</xpExtAddr>
```

```
<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>
```

11. iDigi SMS

11.1 Receiving iDigi 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:
2. Dia Data. The iDigi server 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 Service subscription or not.
 - a. Python programs specify the data contents and, optionally, the file name (called 'path' in `idigisms.send()`).
 - b. If a path is not specified, the file will be stored with a name of `SmsUnnamed` and will be archived (`fdArchive true`).

11.2 Sending iDigi SMS Messages via Web Services

iDigi sends send SMS requests to iDigi devices via SCI. The behavior is very similar to RCI processing from a user's perspective. For more information on the SCI web services interface, please refer to the iDigi Web Services Programming Guide.

iDigi 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 iDigi 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 iDigi 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 iDigi device back to the server for a command. This is primarily intended to allow you to control the number of iDigi 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.

- `cached="true|false|only"`

iDigi 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".

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

11.3 iDigi SMS Command Children

<request_connect>	Request an iDigi data connection. If a connection has already been made, this will force a reconnect of the management session. - No parameters
<reboot>	Reboot device immediately - No parameters
<ping>	Request to determine whether device is able to receive SMS messages - 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 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.
<dia>	Send a Dia command to the running Dia instance. The format of this request is documented in the Dia SMS presentation documentation.
<provision>	Set the iDigi 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

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.
---	---

11.4 Shoulder-Tap iDigi SMS Support

iDigi can be used to send an iDigi SMS request connect to an iDigi device which will cause it to connect back to the server. Once it is connected, web services requests and UI actions can be made to the iDigi device. With this support, iDigi devices no longer need to maintain an iDigi connection at all times. Connections instead can be established dynamically.

This section describes the web services actions to accomplish this. All of these actions can be performed in the iDigi UI as well.

11.4.1 Configure iDigi with the Phone Number of the iDigi Device

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

To provision the phone number of an iDigi device in iDigi, the phone number is added to an entry in NetworkInterface that represents a SIM installed in an iDigi device.

1. Retrieve phone number from iDigi Device

The phone number of the iDigi device can be retrieved using the following RCI request and example response (the phone number is in bold):

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

```
<g3quality max="5">0</g3quality>
<rstat code="1">Registered (Home Network)</rstat>
<cid>34016</cid>
<lac>32004</lac>
<imsi>310410316937398</imsi>
<iccid>89014104243169373988</iccid>
<phnum>18774344439</phnum>
<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>
  <data>5</data>
</varinfo>
<varinfo index="11">
  <desc>Max Power RACH</desc>
  <data>0</data>
</varinfo>
<varinfo index="12">
```

```

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

```

2. Find existing NetworkInterface record to update

Find the NetworkInterface record to update for your iDigi device id. One way to do this is to perform a GET on /ws/DeviceInterface/?condition=devConnectwareId='00000000-00000000-00409DFF-FF2EB94D' (replace with your iDigi device id).

If no NetworkInterface entries are returned, you will need to create a NetworkInterface entry. Use the example below as a guide. See section 8.7 NetworkInterface for more information.

An example reply:

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
    <resultTotalRows>3</resultTotalRows>
    <requestedStartRow>0</requestedStartRow>
    <resultSize>1</resultSize>
    <requestedSize>1000</requestedSize>
    <remainingSize>0</remainingSize>
    <DeviceInterface>
        <id>
            <devId>6</devId>
            <devVersion>0</devVersion>
            <niId>26</niId>
            <niVersion>0</niVersion>
        </id>
        <devRecordStartDate>2011-01-13T18:22:00Z</devRecordStartDate>
        <devMac>00:40:9D:2E:B9:4D</devMac>
        <devCellularModemId>355633002498656</devCellularModemId>
        <devConnectwareId>00000000-00000000-00409DFF-FF2EB94D</devConnectwareId>
        <cstId>10</cstId>
        <grpId>10</grpId>
        <devEffectiveStartDate>2011-01-05T21:37:00Z</devEffectiveStartDate>
        <devTerminated>>false</devTerminated>
        <niRecordStartDate>2011-02-15T21:45:00Z</niRecordStartDate>
        <niInterfaceType>0</niInterfaceType>
        <niEffectiveStartDate>2011-02-15T20:25:00Z</niEffectiveStartDate>
        <niTerminated>>false</niTerminated>
    </DeviceInterface>
</result>

```



Find the niId of the NetworkInterface record to be updated. niId is 26 in the above example.

Retrieve the NetworkInterface record using the id found above: /ws/NetworkInterface/26/0 (0 means most recent version):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<result>
  <resultTotalRows>1</resultTotalRows>
  <requestedStartRow>0</requestedStartRow>
  <resultSize>1</resultSize>
  <requestedSize>1000</requestedSize>
  <remainingSize>0</remainingSize>
  <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>18774344439</niPhone>
    <niPhoneCarrier>3</niPhoneCarrier>
    <niActivePhone>false</niActivePhone>
    <niIdigiPhone>32075</niIdigiPhone>
    <niIdigiServiceId>idgp</niIdigiServiceId>
  </NetworkInterface>
</result>
```

3. Update NetworkInterface Record

Update the NetworkInterface record with the phone number by copying the contents of <NetworkInterface> from the get above, adding niPhone, niActivePhone, and removing ID:

The values added below are:

niActivePhone: true (to indicate this is the active iDigi SMS record. There can be more than one NetworkInterface records per iDigi device. Only one can have niActivePhone true).

niPhone: The phone number of the SIM

PUT /ws/NetworkInterface/26

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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>18774344439</niPhone>
  <niActivePhone>true</niActivePhone>
</NetworkInterface>
```



11.4.2 Configure Device to Receive iDigi SMS Commands

The following example RCI request will configure an iDigi device to enable iDigi SMS, configure iDigi SMS, disable client initiated iDigi connections, and configure paged iDigi 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>idgp</service_identifier>
    </idigisms>
    <mgtconnection index="1">
      <connectionType>client</connectionType>
      <connectionEnabled>off</connectionEnabled>
    </mgtconnection>
    <mgtconnection index="4">
      <connectionType>paged</connectionType>
      <connectionEnabled>on</connectionEnabled>
      <pagedConnectionOverrideEnabled>on</pagedConnectionOverrideEnabled>
      <serverArray index="1">
        <serverAddress>en://my.idigi.com</serverAddress>
      </serverArray>
    </mgtconnection>
    <mgtglobal>
      <connIdleTimeout>2220</connIdleTimeout>
    </mgtglobal>
    <mgtnetwork index="1">
      <networkType>modemPPP</networkType>
      <connectMethod>mt</connectMethod>
      <mtRxKeepAlive>3000</mtRxKeepAlive>
      <mtTxKeepAlive>3000</mtTxKeepAlive>
      <mtWaitCount>3</mtWaitCount>
    </mgtnetwork>
  </set_setting>
</rci_request>
```



11.4.3 RCI for iDigi SMS

RCI group: **idigisms**

Field	Options	Description
state	on, off	iDigi SMS support enabled or disabled. If off, SMS messages will not be processed.
phnum	number	The phone number that the iDigi device will use to send messages to iDigi. This needs to be obtained from Digi (each cluster has its own phone number).
service_identifier	string	An id that when combines with the phone number forms the complete address of the iDigi server. This needs to be obtained from Digi (each cluster has its own phone number).
restrict_sender	on, off	If on, only iDigi SMS messages originating from “phnum” and with the service id “service_identifier” will be honored as iDigi SMS messages.

RCI group: **smscell**

Field	Options	Description
state	on, off	Enables basic SMS support in the iDigi device. This needs to be on for iDigi SMS to communicate.

RCI group: **mgmtconnection index = “1”** (client initiated iDigi connections)

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

RCI group: **mgmtconnection index = “4”** (paged - i.e. temporary - iDigi connections)

Field	Options	Description
connectionEnabled	on, off	Enables temporary connections. A connection request results in a paged iDigi connection being established to the server. If this parameter is off, a connection will not be made.
pagedConnectionOverrideEnabled	on, off	When on, if 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 the iDigi 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 iDigi 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 iDigi SMS Request Connect

To send a connect request to an iDigi device via iDigi 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 iDigi SMS requests to iDigi 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 iDigi. 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 iDigi 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"`
 - This controls whether an iDigi SMS reply is sent by the iDigi device back to the server for a command. This is primarily intended to allow the iDigi SMS user to directly control the number of iDigi 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.
 - all means return a reply iDigi SMS
 - errors means request a reply but the result of the command will only show errors
 - none means do not request a response.

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

`<request_connect>`

Request iDigi 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 iDigi device may be found by performing a GET on `/ws/DeviceCore`.

The result has an entry for each iDigi device. In that entry, the element `dpConnectionStatus` is 0 if the iDigi device is disconnected and 1 if connected:

`<dpConnectionStatus>0</dpConnectionStatus>`

NOTE: A GET on `/ws/DeviceCore` returns a list of all iDigi devices. To retrieve status for a single iDigi device, issue a GET on `/ws/DeviceCore/{id}` where the `id` is the `id` associated with a particular iDigi device.

11.4.6 Send a Disconnect

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

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

Appendix A. Best Practices

A.1 Multiple Queries

When there are multiple things to do they can be wrapped into a single request. For example to get a list of files in python, look up the device info and 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>
    <rci_request version="1.1">
      <!-- Request python files -->
      <do_command target="file_system">
        <ls dir="/WEB/python"/>
      </do_command>
      <!-- Lookup device state -->
      <query_state>
        <device_info/>
      </query_state>
      <!-- Return system settings -->
      <query_setting>
        <system/>
      </query_setting>
    </rci_request>
  </send_message>
</sci_request>
```

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 example use case is as follows:

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 the iDigi
 * session data to avoid authentication overhead.
 *
 * Written for use with a Digi ConnectPort X2/4/8 product connected to
 * developer.idigi.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 the iDigi server for a given web service request
URL url = new URL("http://developer.idigi.com/ws/sci");
URLConnection conn = (URLConnection) 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\"> \r\n");
out.write(" <send_message cache=\"false\"> \r\n");
out.write(" <targets> \r\n");
out.write(" <device id=\"" + deviceId + "\"/> \r\n");
out.write("</targets> \r\n");
out.write(" <rci_request version=\"1.1\"> \r\n");
out.write(" <query_state><device_stats></query_state> \r\n"); out.write("</rci_request> \r\n");
out.write("</send_message> \r\n");
out.write("</sci_request> \r\n");
out.close();
// Get input stream from response and convert to String conn.disconnect();
conn.setDoInput(true);
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 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 is not listed and it has a page that is user defined it will parse the page contents and generate the field itself.

Example for reference:

```
<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>

    <menu id='advanced_cfg' name='Advanced Configuration' page=''
      dataRootDefault='settings' data='' required='false'
      organizeByGroup='true' readonly='false' indexBy=''>

      <automenu page='' dataRootDefault='settings'
        data='settings:*' readonly='false'>
      </automenu>
    </menu>

  </navigation>
</ui>
```

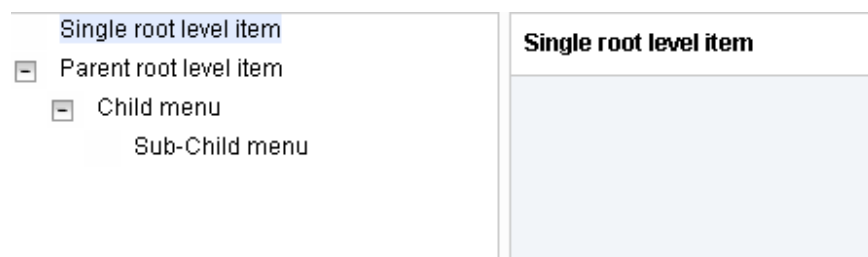
B.1.1 Menu Element

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 an iDigi 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 iDigi properties page. This is the equivalent of creating a page with the contents being just an "<unprocessed/>" element (see below in Page Contents section).

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

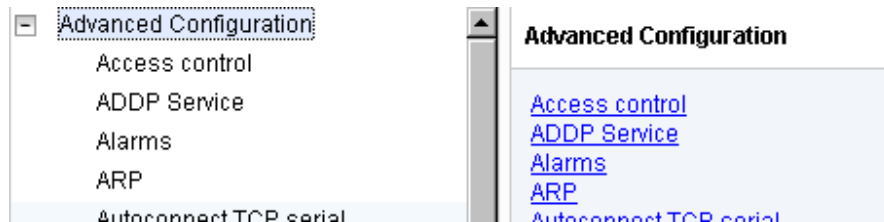
Default root for the data fields when not explicitly specified (optional, settings is default)

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. In example:

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

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

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 xhtml and allows the following tags: b, p, i, s, a, img, table, thead, tbody, tfoot, tr, th, td, dd, dl, dt, em, h1, h2, h3, h4, h5, h6, li, ul, ol, span, div, strike, strong, sub, sup, pre, del, code, blockquote, strike, br, hr, small, big, property, unprocessed, exclude. Most of these tags are standard HTML and will be rendered accordingly in the page area of the device properties when its corresponding menu is selected.

```
<page id='test_page'>
  <b>My iDigi Manager server:</b>
  <property rciId='settings:mgmtconnection/1/serverAddress' />
  <hr />
  <h1>Advanced:</h1>
  <unprocessed>
    <exclude rciId='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 rciId 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 iDigi Manager server:</b>
  <property rciId='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 rciId='settings:mgmtconnection/1/timedConnectionPeriod' />
  </unprocessed>
</page>
```

B.1.4 Help Templates

HTML templates

Example for reference:

```
<ui>
  <content>
    <help id='test_page_help'>
      <b>Help</b>
      <h1 style="color:red">lots of HTML options!</h1>
    </help>
  </content>
</ui>
```

Help templates contain an id attribute that are used as reference. The contents are xhtml that will be displayed in popup when the help is clicked in the properties page.