rainforest automation

EAGLE-200™

Local API Manual

Version 1.0

Sep, 2017

Copyright © 2017 by RAINFOREST AUTOMATION, INC (“RFA”). All rights reserved.

No part of this manual may be reproduced or transmitted in any from without the expressed, written permission of RFA.

Under copyright law, this manual or the software described within, cannot be copied, in whole or part, without the written consent of the manufacturer, except in the normal use of the software to make a backup copy. The same proprietary and copyright notices must be affixed to any permitted copies as were affixed to the original. This exception does not allow copies to be made for others, whether or not sold, but all of the material purchased (with all backup copies) can be sold, given, or loaned to another person. Under the law, copying includes translating into another language or format.

Rainforest Automation may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Rainforest Automation, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Trademarks

Third-party brands and company names mentioned herein may be trademarks and/or registered trademarks of their respective companies and are the sole property of their respective manufacturers.

Notice

The author(s) assumes no responsibility for any errors or omissions that may appear in this document nor does it make a commitment to update the information contained herein.

EAGLE-200™ – Intelligent Control Gateway

RFA-Z114
Version 1.0

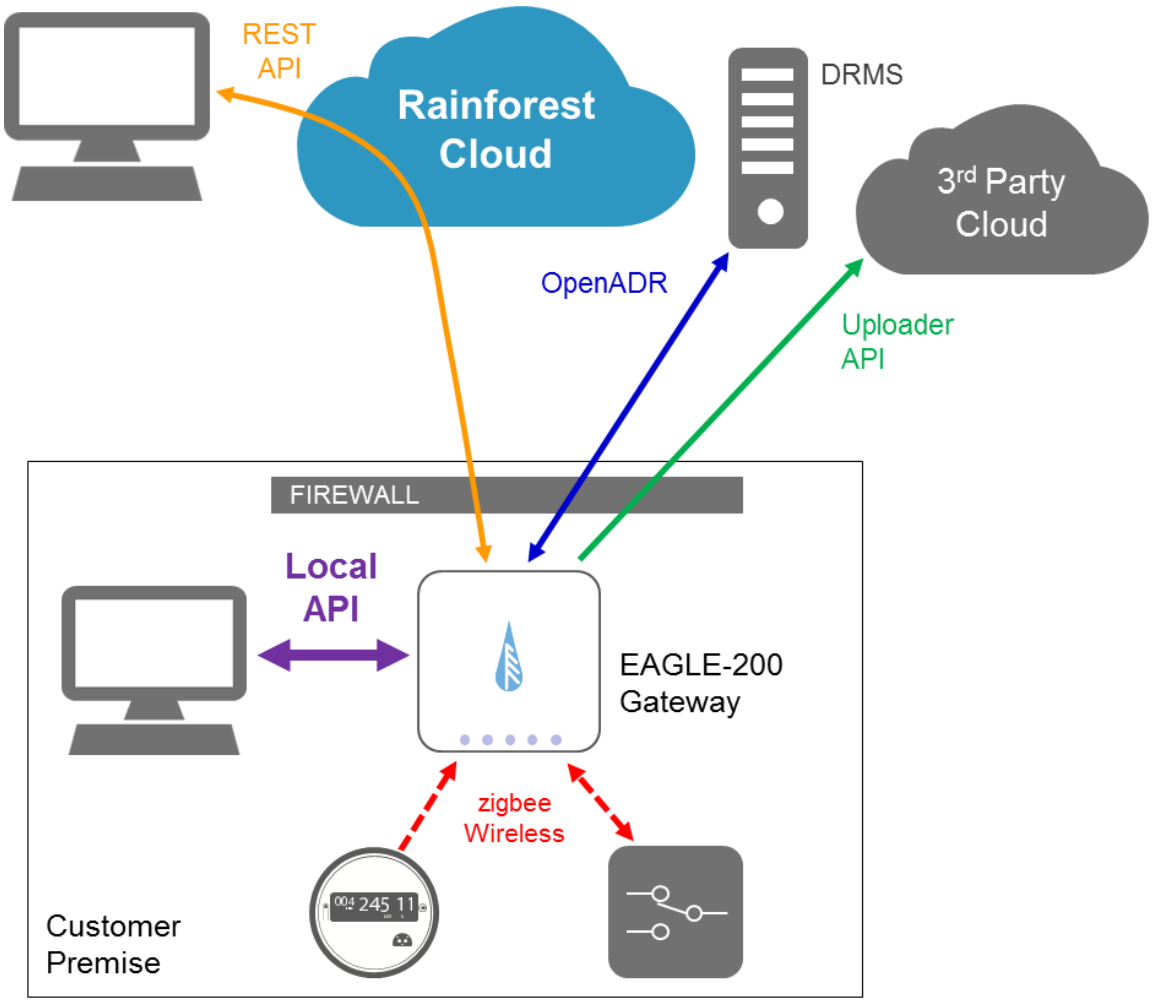
Local API Manual

Table of Contents

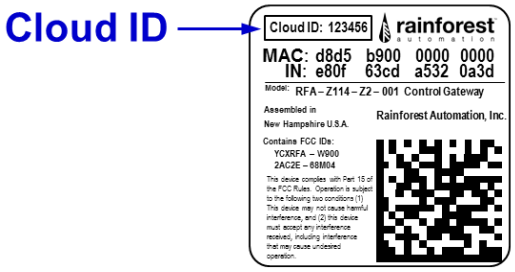
OVERVIEW

The Rainforest EAGLE-200 gateway is a node on an Ethernet network that uses HTTP (Hypertext Transfer Protocol) over TCP/IP (Transmission Control Protocol/Internet Protocol) to communicate with web-enabled entities. The API (Application Programming Interface) described in this document is the mechanism that allows Ethernet devices on the same local network as the EAGLE to issue commands and get data from the EAGLE.

Web-enabled entities outside of the EAGLE's local network (i.e., outside the firewall) should use the *EAGLE-200 REST API* to communicate with the EAGLE.



The EAGLE is identified by using its unique Cloud ID, which is made up of the last 6 digits of its Ethernet MACID. The Cloud ID can be found in the upper left corner of the label on the underside of the EAGLE.



The EAGLE can be found on the local Ethernet network using zeroconf name resolution (also known as mDNS). The name will be in the form “eagle-xxxxxx”, where xxxxxx is the Cloud ID of the EAGLE.

Applications connected to the local network can effectively communicate with the EAGLE using 8-bit Extended ASCII characters (code page 1252) formed into HTTP POST requests. The body of these POST requests contains XML (eXtensible Markup Language) fragments. These XML fragments contain commands for the EAGLE.

DATA FORMAT

Commands

The Application sends commands in HTTP POST requests. POST requests have the following structure:

```
POST <URL> HTTP/1.0
<headers>
<blank>
<body>
```

Where:

- Every line ends with the carriage return and line feed characters (0x0D, 0x0A).
- `<URL>` is the Uniform Resource Locator (web address) of the EAGLE.
- `<headers>` are a variable number of HTTP headers; each header is on its own line. The following items must be included in the headers:
  - `Content-type: text/xml`
  - `Content-Length: xx`
    where "xx" is the number of characters in the body of the POST
  - `Authorization: Basic xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx`
    where the 32-character Basic Authentication credential is formed using the EAGLE Cloud ID as the username, and the EAGLE Install Code as the password.
- `<blank>` is a blank line, consisting only of the carriage return and line feed characters (0x0D, 0x0A).
- `<body>` is the main text of the POST request, which has the structure shown below.

The body of the POST consists of XML Fragments. An XML Fragment is a stripped-down XML Element. The EAGLE uses XML Fragments to simplify the parsing of the data stream, while providing a data structure that is flexible and human readable.

The XML Fragments have the following structure:

```
<tag>
      <element>value</element>
      …
</tag>
```

Where:

- Every line ends with the carriage return and line feed characters (0x0D, 0x0A).
- `<tag>` is the start tag for the XML Fragment; each type will have a unique tag name.

- `<element>` is the start tag for an element; there will be one or more child elements in the fragment; each element will have a unique element name.
- … indicates the variable number of specific elements.

Element values can be of various types:

- `{string}` indicates an element consisting of Extended ASCII text
- `{enumeration}` indicates an element that can have a specific list of values.
- `0xFFFFF` indicates an element consisting of a base16 (hex) number
- `00` indicates an element consisting of an integer
- `000.000` indicates an element consisting of a signed decimal number

`[<element>]` – square brackets indicate optional elements.
`value1|value2|value3` – vertical bars separate valid values in an enumeration list.

Note that element names are case insensitive; the case is used strictly for legibility. The EAGLE will ignore case when receiving POST requests.

Example

Here is an example of a POST request:

```
POST http://192.168.100.164/cgi-bin/post_manager HTTP/1.0
Content-type: text/xml
Content-Length: 43
Authorization: Basic MDA0NzkyOmJmYjBmYzA1ZjUxYTM5MzI=

<Command><Name>wifi_status</Name></Command>
```

Responses

The EAGLE generates a valid HTTP response to each POST request. These look like:

```
HTTP/1.0 <code>
<headers>
<blank>
<body>
```

Where:

- Every line ends with the carriage return and line feed characters (0x0D, 0x0A).
- `<code>` is an HTTP status code, which consists of a 3-digit number and a short text phrase. This is usually "200 OK".
- `<headers>` are a variable number of HTTP headers; each header is on its own line.
- `<blank>` is a blank line, consisting only of the carriage return and line feed characters (0x0D, 0x0A).
- `<body>` is the main text of the response, consisting of XML fragments.

Example

Here is an example of a response:

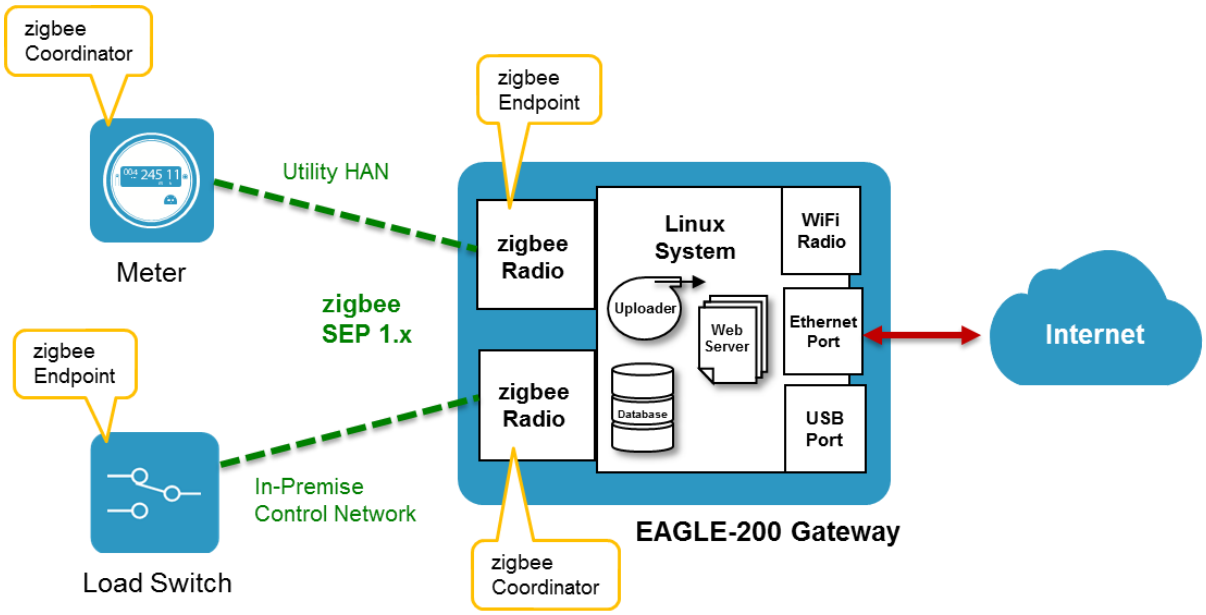
```
HTTP/1.0 200 OK
Date: Sat, 19 Aug 2017 01:04:06 GMT
Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: Authorization
Content-Length: 205
Content-Type: text/html

<WiFiStatus>
  <Enabled>Y</Enabled>
  <Type>router</Type>
  <SSID>eagle-004792 (router)</SSID>
  <Encryption>WPA2 PSK (CCMP)</Encryption>
  <Channel>11</Channel>
  <IpAddress>192.168.7.1</IpAddress>
</WiFiStatus>
```

HTTP 1.1

An HTTP POST can also specify that HTTP 1.1 be used. In this case, the reply generated by the EAGLE will contain the header element “Transfer-Encoding: chunked” in place of the “Content-Length” header element. This indicates that the body of the response will be divided into chunks. Each chunk of data will be preceded by the chunk length on a line by itself as a hexadecimal number. Each chunk will also be followed by a blank line. The end of the message will be indicated by a zero on a line by itself.

SUBDEVICE NETWORKS

The Rainforest EAGLE-200 gateway contains two zigbee radios that support two independent zigbee wireless networks:



Utility HAN: The first zigbee radio in the EAGLE communicates directly with a smart meter equipped with zigbee wireless capability that supports the Smart Energy Profile (SEP) protocol standard (almost all US smart meters support this standard). This is called the Home Area Network (HAN). It consists of the utility smart meter, which is the network Coordinator, and one or more Endpoints. The EAGLE acts as an Endpoint on the HAN. The EAGLE sees the utility meter as an "electric_meter" device.

Control Network: The second zigbee radio in the EAGLE acts as a zigbee network Coordinator and forms its own independent zigbee network. This network is completely separate from the Utility HAN, and does not connect to the meter. The Control Network allows the EAGLE to communicate with and control a number of zigbee subdevices. These subdevices can use either SEP or Home Automation (HA) zigbee protocols. Subdevices must have been tested by Rainforest Automation to ensure compatibility with the EAGLE. Each supported subdevice has a specific profile loaded on the EAGLE. A list of supported subdevices is available from Rainforest Automation.

The EAGLE can operate with either or both of these networks active. If there is no zigbee smart meter present, then there will be no HAN, but this does not affect the Control Network. Likewise, the EAGLE can read a meter without there being a Control Network. However, the utility meter (if present) and controlled subdevices all share the same Local API.

READING METER DATA

If a zigbee smart meter is present, it will be provisioned by the utility, and the EAGLE will automatically connect to it. This is different from the Control Network, where the EAGLE must be provisioned through the API, and each subdevice is thereby added to the network.

To get data from the meter, you must first discover its Hardware Address. This can be done by issuing a "device_list" command to the EAGLE through the Local API, which is done by sending a POST with the following text in the body:

```
<Command>
    <Name>device_list</Name>
</Command>
```

The `<DeviceList>` response from the EAGLE will contain a list of all the devices that are in its device tables – from both the HAN and the Control Network. Each device is a `<Device>` element. The text for the meter listing will be similar to the following:

```
<DeviceList>
      ⋮
    <Device>
        <HardwareAddress>0x000781000081fd0b</HardwareAddress>
        <Manufacturer>generic</Manufacturer>
        <ModelId>electric_meter</ModelId>
        <Protocol>Zigbee</Protocol>
        <LastContact>0x5989f8f5</LastContact>
        <ConnectionStatus>Connected</ConnectionStatus>
        <NetworkAddress>0x0000</NetworkAddress>
    </Device>
      ⋮
</DeviceList>
```

You can then use the 16-digit hexadecimal `<HardwareAddress>` element from the meter listing to issue a "device_query" command, as follows:

```
<Command>
    <Name>device_query</Name>
    <DeviceDetails>
        <HardwareAddress>0x000781000081fd0b</HardwareAddress>
    </DeviceDetails>
    <Components>
        <Component>
            <Name>Main</Name>
            <Variables>
                <Variable>
                    <Name>zigbee:InstantaneousDemand</Name>
                </Variable>
            </Variables>
        </Component>
    </Components>
</Command>
```

The EAGLE will respond with a <Device> response containing the value of the requested variable (in this case instantaneous demand) that it has stored in its data buffer for this device:

```
<Device>
    <DeviceDetails>
        <Name>Power Meter</Name>
        <HardwareAddress>0x000781000081fd0b</HardwareAddress>
        <Protocol>Zigbee</Protocol>
        <NetworkAddress>0x0000</NetworkAddress>
        <Manufacturer>Generic</Manufacturer>
        <ModelId>electric_meter</ModelId>
        <LastContact>0x59a0b67c</LastContact>
        <ConnectionStatus>Connected</ConnectionStatus>
    </DeviceDetails>
    <Components>
        <Component>
            <HardwareId></HardwareId>
            <FixedId>0</FixedId>
            <Name>Main</Name>
            <Variables>
                <Variable>
                    <Name>zigbee:InstantaneousDemand</Name>
                    <Value>21.499 kW</Value>
                </Variable>
            </Variables>
        </Component>
    </Components>
</Device>
```

Note that the <Value> element is an ASCII text field. To get the numerical value you will need to convert to a number.

To get a list of all of the variables available from the meter, you can issue a “device_details” command:

```
<Command>
    <Name>device_details</Name>
    <DeviceDetails>
      <HardwareAddress>0x000781000081fd0b</HardwareAddress>
    </DeviceDetails>
</Command>
```

The EAGLE <Device> response looks like this:

```
<Device>
    <DeviceDetails>
        <Name>Power Meter</Name>
        <HardwareAddress>0x000781000081fd0b</HardwareAddress>
        <Protocol>Zigbee</Protocol>
        <Manufacturer>Generic</Manufacturer>
        <ModelId>electric_meter</ModelId>
    </DeviceDetails>
```

```
    <Components>
        <Component>
            <Name>Main</Name>
            <FixedId>0</FixedId>
            <Variables>
                <Variable>zigbee:InstantaneousDemand</Variable>
                <Variable>zigbee:Multiplier</Variable>
                <Variable>zigbee:Divisor</Variable>
                <Variable>zigbee:CurrentSummationDelivered</Variable>
                <Variable>zigbee:Price</Variable>
                <Variable>zigbee:RateLabel</Variable>
                <Variable>zigbee:Message</Variable>
            </Variables>
        </Component>
    </Components>
</Device>
```

The values of these variables can be requested simply by listing them in the `<Variables>` list in the “device_query” command.

A shortcut to get all of the variable values is to use the `<All>Y</All>` component:

```
<Command>
    <Name>device_query</Name>
    <DeviceDetails>
        <HardwareAddress>0x000781000081fd0b</HardwareAddress>
    </DeviceDetails>
    <Components>
       <All>Y</All>
    </Components>
</Command>
```

MONITORING AND CONTROLLING A SMART PLUG

To control and monitor a smart plug on the Control Network, the EAGLE must first be provisioned with the credentials of the smart plug. This is done by using the "device_add" command:

```
<Command>
    <Name>device_add</Name>
    <DeviceDetails>
        <HardwareAddress>0x00244600000ebaba</HardwareAddress>
        <InstallCode>0x0a1b2c3d4e5f6a7b</InstallCode
        <Manufacturer>SafePlug</Manufacturer>
        <ModelId>1202</ModelId>
        <Protocol>Zigbee</Protocol>
        <Name>safeplug 1202</Name>
    </DeviceDetails>
    <NetworkInterface>
        <HardwareAddress>0xd8d5b9000000b49f</HardwareAddress>
    </NetworkInterface>
</Command>
```

Where:

- `<HardwareAddress>` element in the `<DeviceDetails>` component is the 16-digit hexadecimal MAC Address of the smart plug.
- `<InstallCode>` is the 16-digit hexadecimal Install Code of the smart plug.
- `<HardwareAddress>` element in the `<NetworkInterface>` component is the 16-digit hexadecimal MAC Address of the EAGLE.

Note that only subdevices that have a device profile programmed into the EAGLE can be added to that EAGLE's Control Network. The `<Manufacturer>` and `<ModelId>` elements in the "device_add" command must match the entries in the profile exactly. See the Appendix for a list of supported devices.

Once a subdevice has been added to the EAGLE Control Network, it will appear in the list of devices for that EAGLE. So, the response to a "device_list" command (see previous section) will have the following `<Device>` element:

```
<DeviceList>
      ⋮
    <Device>
        <HardwareAddress>0x00244600000ebaba</HardwareAddress>
        <Manufacturer>SafePlug</Manufacturer>
        <ModelId>1202</ModelId>
        <Protocol>Zigbee</Protocol>
        <LastContact>0x5989f8f5</LastContact>
        <ConnectionStatus>Connected</ConnectionStatus>
        <NetworkAddress>0x20db</NetworkAddress>
    </Device>
      ⋮
</DeviceList>
```

In this example, we are using a SafePlug 1202 smart plug. Unlike the meter shown in the previous section, it has three components: “Global”, “Receptacle 1”, and “Receptacle 2”. This can be seen in the response to the “device_details” command:

```
<Device>
    <DeviceDetails>
        <Name>1202</Name>
        <HardwareAddress>0x00244600000ebaba</HardwareAddress>
        <Protocol>Zigbee</Protocol>
        <Manufacturer>SafePlug</Manufacturer>
        <ModelId>1202</ModelId>
    </DeviceDetails>
    <Components>
        <Component>
            <Name>Global</Name>
            <FixedId>0</FixedId>
            <Variables>
                <Variable>zigbee:ZclVersion</Variable>
                <Variable>zigbee:ApplicationVersion</Variable>
                <Variable>zigbee:StackVersion</Variable>
                <Variable>zigbee:HwVersion</Variable>
                <Variable>zigbee:Manufacturer</Variable>
                <Variable>zigbee:ModelId</Variable>
                <Variable>zigbee:DateCode</Variable>
                <Variable>zigbee:PowerSource</Variable>
                <Variable>zigbee:SWBuildID</Variable>
                <Variable>zigbee:UtcTime</Variable>
                <Variable>zigbee:LocalTime</Variable>
            </Variables>
        </Component>
        <Component>
            <Name>Receptacle 1</Name>
            <FixedId>1</FixedId>
            <Variables>
                <Variable>safeplug:RightplugState</Variable>
                <Variable>safeplug:RightplugUid</Variable>
                <Variable>safeplug:RightplugMid</Variable>
                <Variable>safeplug:RightplugPid</Variable>
                <Variable>safeplug:RightplugLid</Variable>
                <Variable>safeplug:RightplugVid</Variable>
                <Variable>safeplug:RightplugRatings</Variable>
                <Variable>safeplug:RightplugWattHours</Variable>
                <Variable>safeplug:EfciState</Variable>
                <Variable>safeplug:EfciLevel</Variable>
                <Variable>safeplug:EfciVnow</Variable>
                <Variable>safeplug:EfciInow</Variable>
                <Variable>safeplug:EfciPnow</Variable>
                <Variable>zigbee:OnOff</Variable>
                <Variable>zigbee:InstantaneousDemand</Variable>
                <Variable>zigbee:SummationDelivered</Variable>
                <Variable>zigbee:SummationReceived</Variable>
                <Variable>zigbee:DeviceClass</Variable>
                <Variable>zigbee:UnitofMeasure</Variable>
                <Variable>zigbee:Multiplier</Variable>
```

```
                <Variable>zigbee:Divisor</Variable>
                <Variable>zigbee:DemandFormatting</Variable>
            </Variables>
        </Component>
        <Component>
            <Name>Receptacle 2</Name>
            <FixedId>2</FixedId>
            <Variables>
                <Variable>safeplug:RightplugState</Variable>
                <Variable>safeplug:RightplugUid</Variable>
                <Variable>safeplug:RightplugMid</Variable>
                <Variable>safeplug:RightplugPid</Variable>
                <Variable>safeplug:RightplugLid</Variable>
                <Variable>safeplug:RightplugVid</Variable>
                <Variable>safeplug:RightplugRatings</Variable>
                <Variable>safeplug:RightplugWattHours</Variable>
                <Variable>safeplug:EfciState</Variable>
                <Variable>safeplug:EfciLevel</Variable>
                <Variable>safeplug:EfciVnow</Variable>
                <Variable>safeplug:EfciInow</Variable>
                <Variable>safeplug:EfciPnow</Variable>
                <Variable>zigbee:OnOff</Variable>
                <Variable>zigbee:InstantaneousDemand</Variable>
                <Variable>zigbee:SummationDelivered</Variable>
                <Variable>zigbee:SummationReceived</Variable>
                <Variable>zigbee:DeviceClass</Variable>
                <Variable>zigbee:UnitofMeasure</Variable>
                <Variable>zigbee:Multiplier</Variable>
                <Variable>zigbee:Divisor</Variable>
                <Variable>zigbee:DemandFormatting</Variable>
            </Variables>
        </Component>
    </Components>
</Device>
```

The “Global” component contains variables common to the entire unit, while the “Receptacle 1” component is specific to the upper outlet of the plug, and “Receptacle 2” is specific to the lower outlet.

Monitoring

Just like the meter, you can issue a “device_query” command to get the value of any variable that the EAGLE has in its data buffer for this device.

```
<Command>
    <Name>device_query</Name>
    <DeviceDetails>
        <HardwareAddress>0x00244600000ebaba</HardwareAddress>
    </DeviceDetails>
    <Components>
        <Component>
            <Name>Receptacle 1</Name>
            <Variables>
```

```
                <Variable>
                    <Name>zigbee:InstantaneousDemand</Name>
                </Variable>
            </Variables>
        </Component>
        <Component>
            <Name>Receptacle 2</Name>
            <Variables>
                <Variable>
                    <Name>zigbee:InstantaneousDemand</Name>
                </Variable>
            </Variables>
        </Component>
    </Components>
</Command>
```

This will return the Instantaneous Demand value for each of the outlets in the plug.

```
<Device>
    <DeviceDetails>
        <Name>1202</Name>
        <HardwareAddress>0x00244600000ebaba</HardwareAddress>
        <Protocol>Zigbee</Protocol>
        <NetworkAddress>0x78fe</NetworkAddress>
        <Manufacturer>SafePlug</Manufacturer>
        <ModelId>1202</ModelId>
        <LastContact>0x59a096b4</LastContact>
        <ConnectionStatus>Connected</ConnectionStatus>
    </DeviceDetails>
    <Components>
        <Component>
            <HardwareId>0x0b</HardwareId>
            <FixedId>1</FixedId>
            <Name>Receptacle 1</Name>
            <Variables>
                <Variable>
                    <Name>zigbee:InstantaneousDemand</Name>
                    <Value>0.120 kW</Value>
                </Variable>
            </Variables>
        </Component>
        <Component>
            <HardwareId>0x0c</HardwareId>
            <FixedId>2</FixedId>
            <Name>Receptacle 2</Name>
            <Variables>
                <Variable>
                    <Name>zigbee:InstantaneousDemand</Name>
                    <Value>0.000 kW</Value>
                </Variable>
            </Variables>
        </Component>
    </Components>
</Device>
```

Note that the `<Value>` elements are ASCII text fields. To get the numerical values you will need to convert to numbers.

Control

The outlets of the plug can be controlled using a "device_control" command, which has a syntax similar to the "device_query" command. For example, the following sequence will turn off the top outlet, and turn on the bottom outlet of the plug:

```
<Command>
    <Name>device_control</Name>
    <DeviceDetails>
        <HardwareAddress>0x00244600000ebaba</HardwareAddress>
    </DeviceDetails>
    <Components>
        <Component>
            <Name>Receptacle 1</Name>
            <Variables>
                <Variable>
                    <Name>zigbee:OnOff</Name>
                    <Value>off</Value>
                </Variable>
            </Variables>
        </Component>
        <Component>
            <Name>Receptacle 2</Name>
            <Variables>
                <Variable>
                    <Name>zigbee:OnOff</Name>
                    <Value>on</Value>
                </Variable>
            </Variables>
        </Component>
    </Components>
</Command>
```

To check the on/off status of the outlets of the plug, you can issue a "device_query" command:

```
<Command>
    <Name>device_query</Name>
    <DeviceDetails>
        <HardwareAddress>0x00244600000ebaba</HardwareAddress>
    </DeviceDetails>
    <Components>
        <Component>
            <Name>Receptacle 1</Name>
            <Variables>
                <Variable>
                    <Name>zigbee:OnOff</Name>
                </Variable>
            </Variables>
        </Component>
        <Component>
```

```
                <Name>Receptacle 2</Name>
                <Variables>
                    <Variable>
                        <Name>zigbee:OnOff</Name>
                    </Variable>
                </Variables>
            </Component>
        </Components>
    </Command>
```

The status values in the response will be those stored in the EAGLE data buffers.

To update the buffers, you can first issue a “device_query” command with an additional `<Refresh>Y</Refresh>` element in the variable:

```
                    ⋮
                <Variable>
                    <Name>zigbee:OnOff</Name>
                    <Refresh>Y</Refresh>
                </Variable>
                    ⋮
```

This will trigger the EAGLE to send a request to the plug to get the status of this outlet. Note that a “device_query” command with refresh does not receive a value for the refreshed variable. Another “device_query” command needs to be sent to get the value (it may take some time to update the buffer).

MONITORING AND CONTROLLING A THERMOSTAT

Just like a smart plug, a thermostat must first be added the Control Network before it can be seen by the EAGLE. Here is the "device_add" command for an Emerson EE542-1Z Smart Energy Thermostat:

```
<Command>
    <Name>device_add</Name>
    <DeviceDetails>
        <HardwareAddress>0x00244600000abcde</HardwareAddress>
        <InstallCode>0x0a1b2c3d4e5f6a7b</InstallCode
        <Manufacturer>emerson</Manufacturer>
        <ModelId>ee542</ModelId>
        <Protocol>Zigbee</Protocol>
        <Name>emerson_ee542</Name>
    </DeviceDetails>
    <NetworkInterface>
        <HardwareAddress>0xd8d5b9000000b49f</HardwareAddress>
    </NetworkInterface>
</Command>
```

Where:

- `<HardwareAddress>` element in the `<DeviceDetails>` component is the 16-digit hexadecimal MAC Address of the thermostat.
- `<InstallCode>` is the 16-digit hexadecimal Install Code of the thermostat.
- `<HardwareAddress>` element in the `<NetworkInterface>` component is the 16-digit hexadecimal MAC Address of the EAGLE.
- `<Manufacturer>` and `<ModelId>` elements must exactly match the entries in the profile for this device (see the Appendix).

Now, the response to a "device_list" command will contain the following `<Device>` element:

```
<DeviceList>
      ⋮
    <Device>
        <HardwareAddress>0x00244600000abcde</HardwareAddress>
        <Manufacturer>emerson</Manufacturer>
        <ModelId>ee542</ModelId>
        <Protocol>Zigbee</Protocol>
        <LastContact>0x5989f5f1</LastContact>
        <ConnectionStatus>Not joined</ConnectionStatus>
        <NetworkAddress>0xffff</NetworkAddress>
    </Device>
      ⋮
</DeviceList>
```

The “device_details” command will give you a list of all the variables available from this thermostat:

```
<Device>
    <DeviceDetails>
        <Name>ee542</Name>
        <HardwareAddress>0x00244600000abcde</HardwareAddress>
        <Protocol>Zigbee</Protocol>
        <Manufacturer>emerson</Manufacturer>
        <ModelId>ee542</ModelId>
    </DeviceDetails>
    <Components>
        <Component>
            <Name>Global</Name>
            <FixedId>0</FixedId>
            <Variables>
                <Variable>zigbee:ZclVersion</Variable>
                <Variable>zigbee:ApplicationVersion</Variable>
                <Variable>zigbee:StackVersion</Variable>
                <Variable>zigbee:HwVersion</Variable>
                <Variable>zigbee:Manufacturer</Variable>
                <Variable>zigbee:PowerSource</Variable>
                <Variable>zigbee:LocalTemperature</Variable>
                <Variable>zigbee:OccupiedCoolingSetpoint</Variable>
                <Variable>zigbee:OccupiedHeatingSetpoint</Variable>
                <Variable>zigbee:SystemMode</Variable>
            </Variables>
        </Component>
    </Components>
</Device>
```

Monitoring

You can issue a “device_query” command to get the temperature read by the thermostat:

```
<Command>
    <Name>device_query</Name>
    <DeviceDetails>
        <HardwareAddress>0x00244600000abcde</HardwareAddress>
    </DeviceDetails>
    <Components>
        <Component>
            <Name>Global</Name>
            <Variables>
                <Variable>
                    <Name>zigbee:LocalTemperature</Name>
                </Variable>
            </Variables>
        </Component>
    </Components>
</Command>
```

The EAGLE will respond with the value in its data buffer:

```
<Device>
    <DeviceDetails>
        <Name>ee542</Name>
        <HardwareAddress>0x00244600000abcde</HardwareAddress>
        <Protocol>Zigbee</Protocol>
        <NetworkAddress>0xf67b</NetworkAddress>
        <Manufacturer>emerson</Manufacturer>
        <ModelId>ee542</ModelId>
        <LastContact>0x598a55a5</LastContact>
        <ConnectionStatus>Connected</ConnectionStatus>
    </DeviceDetails>
    <Components>
        <Component>
            <HardwareId>0x02</HardwareId>
            <FixedId>0</FixedId>
            <Name>Global</Name>
            <Variables>
                <Variable>
                    <Name>zigbee:LocalTemperature</Name>
                    <Value>27.22 C</Value>
                </Variable>
            </Variables>
        </Component>
    </Components>
</Device>
```

Note that the `<Value>` element is an ASCII text field that indicates the scale used (C or F). To get the numerical value you will need to convert to a number.

Just like a smart plug, you can force the EAGLE to send a request to the thermostat to get an updated reading by issuing a “device_query” command with refresh:

```
    ⋮
<Variable>
    <Name>zigbee:LocalTemperature</Name>
    <Refresh>Y</Refresh>
</Variable>
    ⋮
```

You can then issue another “device_query” command to get the updated value.

In addition to the measured room temperature, you can also read the setpoints for heating and cooling, as well as the operational mode.

The mode can have the following values:

Value	Mode
0	Off
1	Auto
2	
3	Cool

4	Heat
5	Emergency Heat
6	Precool
7	Fan Only
8	Dry
9	Sleep

Control

The setpoints and the mode of the thermostat can be set using a “device_control” command. For example, this command will put the thermostat in Cool Mode and set the Cooling Setpoint to 20°C:

```
<Command>
    <Name>device_control</Name>
    <DeviceDetails>
        <HardwareAddress>0x00244600000abcde</HardwareAddress>
    </DeviceDetails>
    <Components>
        <Component>
            <Name>Global</Name>
            <Variables>
                <Variable>
                    <Name>zigbee:OccupiedCoolingSetpoint</Name>
                    <Value>20</Value>
                </Variable>
                <Variable>
                    <Name>zigbee:SystemMode</Name>
                    <Value>3</Value>
                </Variable>
            </Variables>
        </Component>
    </Components>
</Command>
```

Note that temperature values are in degrees Celcius.

APPENDIX: Supported Devices

Type	Brand	Model	Profile elements
Thermostat	RTCOA	CT32	<Manufacturer>RTCOA</Manufacturer> <ModelId>CT32</ModelId>
Thermostat	Carrier	ComfortChoice Touch	<Manufacturer>Carrier</Manufacturer> <ModelId>TouchPCT</ModelId>
Thermostat	Emerson	EE542-1Z	<Manufacturer>emerson</Manufacturer> <ModelId>ee542</ModelId>
Thermostat	ecobee	EB-SmartSi-01	<Manufacturer>Ecobee</Manufacturer> <ModelId>EBSmartSi1</ModelId>
Thermostat	Zen	Zen-01	<Manufacturer>mmbnetworks</Manufacturer> <ModelId>zen-01</ModelId>
Load Switch	Cooper	LCR-6600	<Manufacturer>cooper</Manufacturer> <ModelId>lcr6200</ModelId>
Load Switch	SafePlug	1313	<Manufacturer>SafePlug</Manufacturer> <ModelId>1313</ModelId>
Load Switch	Energate	LC2200	<Manufacturer>energate</Manufacturer> <ModelId>lc2200</ModelId>
Smart Plug	SafePlug	1202	<Manufacturer>SafePlug</Manufacturer> <ModelId>1202</ModelId>
Smart Plug	Energate	PLM6193	<Manufacturer>energate</Manufacturer> <ModelId>plm6193</ModelId>
H/W Switch	Emerson	75A01	<Manufacturer>Rainforest Automation</Manufacturer> <ModelId>Z140E</ModelId>
CT Meter	Neurio	PowerBlaster	<Manufacturer>Neurio</Manufacturer> <ModelId>PB1</ModelId>
Inverter	SolarEdge	SE1000-CCG	<Manufacturer>SolarEdge</Manufacturer> <ModelId>SE1000-CCG</ModelId>