



PulseWorx Gateway Dev-Kit

01-April-2017



19201 Parthenia Street, Suite J
Northridge, CA 91234

P: 818.701.9831

F: 818.701.1506

pcssales@pcslighting.com

www.pcslighting.com



Proper Preparation Prevents Poor Performance

Before beginning, have available these items for review. They can be downloaded from the PCS Support web site resources section as PDF files.

Other documentation section

- UPB Description Document
- PIM Description Document V1.7
- Gateway Firmware Design
- UPStart Export File V5.2
- PulseWorx-App Sources zip file. In the zip file is a SourceReadMe.txt file and two other zip files. Review the SourceReadMe.txt file before working with the 2 source zip files.

Interfaces PulseWorx Gateway section

- Gateway User Guide

PulseWorx App Notes section

- PulseWorx App Note 126: Using the PIM from your own application to send and receive UPB messages
- PulseWorx App Note 127: Using the PIM-IP with your own applications
- PulseWorx App Note 128: Using the PulseWorx Gateway with your own applications

Introduction for those familiar with the serial or USB PIM

If you are already familiar with working with the PIM – sending and receiving commands – as well as the UPB command message format then you already have a good start on the Gateway.

I suggest you first review the Gateway User Guide. This will give you an overview of the capabilities of the Gateway so you have some context for what follows.

When converting from working with the PIM or PIM-IP to working with the Gateway, these general areas are important to focus on:

1. Connecting to the Gateway. There is a connection protocol that must be implemented for initial connection and username / password passing
2. Packaging PIM messages into Gateway messages and sending to the Gateway for onward shipment to the PIM
3. Decoding Gateway messages in general
4. Reading the state table and syncing the application state to the network state
5. Processing the Gateway state update message
6. Handling disconnection when a pulse mode application connects to the Gateway.

One of the big changes is that applications converting from the PIM to the Gateway no longer need to process UPB messages if all they are doing is using those messages to keep device state up to date.

The Gateway still send the messages to your application but you no longer need to process them. The Gateway already determines the effect of the commands and sends state update messages.

Of course, if your application is sending UPB commands to do other things – read and write device configuration memory for example and not just controlling devices – then the application would of course still have to process those UPB message replies.

The last major change, should applications decide to support it, is to give users the ability to view, modify, and save the schedules executed by the Gateway.

PCS has a Windows application (PulseWorx-App or PWA) that works with the Gateway in much the same way as the PCS iOS and Android applications do. A full set of sources are available (C++ with MFC) on the PCS Support web site. This file refers to various pieces of that implementation. This application also has several good uses: It can serve as a reference application so you can compare what your app is doing with what PWA does to check for correctness. It also it has many diagnostics tools that you can use to look at tables stored in the Gateway or to listen in on the message traffic between PWA and the gateway to see what it is sending and receiving.

What is the PulseWorx Gateway?

The PulseWorx Gateway is a replacement for the serial/USB PIM and the PIM-IP that has several advantages over these older devices.

- Multiple connections are supported. If too many connections are attempted a “too many” error is returned.
- Provides a consistent protocol for retrieval and storage of files and tables
- Is discoverable from an application that can send and received UDP messages
- Manages connections made by pulse mode and command mode applications. Only one pulse mode application at a time is allowed and connected message mode applications are terminated when an application enters pulse mode.
- UPB messages received from the powerline are sent to all connected clients. Transmissions from one client are echoed to other connected clients.
- Internally keeps track of the state of the UPB network based upon UPB messages received whether connected to an application or not. Upon application connection it can sync with the current network state by reading this table. When connected, state update messages are sent to clients in terms of device, channel, level.
- Implements a security model with up to 4 users with a user name and password. Privileges are assigned to each user that controls their ability to modify the configuration of the Gateway.
- Contains an autonomous scheduler with time and sun time based schedule entries. Up to 4 schedules are supported with facilities to switch between schedules.

NOTE: The replacement for the older PIMs is called the “PulseWorx Gateway” or simply the “Gateway”. In some documentation it may be referred to as the “PIM-IP2”. Inside the Gateway enclosure are two logical parts. One part is the network side implemented in a Digi-Connect module which executes PCS written firmware. The other part is the PIM that interfaces to the powerline. The PIM part of the Gateway contains the same firmware as a RS232 connected PIM. In the documentation, “PIM” is the UPB interface part of the Gateway. The Digi-Connect side of the Gateway is sometimes referred to as the “Gateway” (a bit ambiguously) or as the “Gateway firmware” or just “the firmware”.

Even though both the Gateway and the PIM-IP have a network connection, the Gateway is not compatible with the existing PIM-IP. Applications that implemented the PIM-IP require software changes to work with the new Gateway.

Before reviewing any further documentation, it is important to understand that the same facilities provided by the PIM are exposed unmodified to implementers of the Gateway. No attempt has been made to craft a higher-level view of a UPB network. Just as applications that interfaced with a UPB network using a serial PIM had to understand the PIM command set and how to encode UPB messages, so too will applications that interface with the Gateway.

PulseWorxApp – Testing and Reference application

To provide a sample for app creators, PCS created a windows application that can be used as a sample for working with the PulseWorx Gateway. This sample application:

- Connects to the PulseWorx Gateway
- Loads tables including the export.uep file from the Gateway memory
- Syncs its internal state with the state tables from the Gateway and the loaded network
- Displays a user interface to control devices and activate/deactivate links
- On a mouse double-click, or a touch screen long tap, displays a popup keypad whose buttons show the same state as the LEDs on the actual keypads if known. The single-click action for the button is performed when the popup keypad button is clicked or with a touchscreen short tap
- Has development features that allow the various features of the Gateway interface to be tested
- Contains a UI for modification of the user table for adding and deleting users, and a complete scheduler.
- Provides a viewer to show all the messages passed between the application and the Gateway

- Provides a viewer that shows Gateway diagnostic messages

The sources of this application are contained in a zip file available for download. It is in Microsoft Visual Studio format.

PulseWorx-App serves three purposes:

First it provides a clear user oriented view of a UPB network – by rooms, devices, and scenes – and allows for simple control. While this could be the model for mobile applications it need not be. If the basic operations to the user are available for device view and control then the presentation could be different.

The second use of this application is to provide a platform for testing many of the Gateway features that applications use – reading and writing files, updating tables, setting the Gateway date and time, etc.

And finally, it can be used from any Windows computer to access the Gateway for control or configuration.

It is hoped that developers will use this application to do “A to B” comparisons between it and their application. This allows them to verify that what their application does is correct.

To enable “Developer Mode” after installing the PulseWorx-App, make sure that you run it at least once then exit. Start Windows RegEdit and go here:

```
HKEY_CURRENT_USER \ Software \ PCSLighting \ PulseWorxApp \ UserInterface
```

Look for the key “DeveloperMode” and set its value to 1.

Exit RegEdit and restart PulseWorx-App and the “Devbelopment” ribbon category will be present.

Using PulseWorx-App

Before going into a roadmap of the app sources, it is best to work with the app to get a feel for what it does. To do this, first begin with UPStart.

After powering on the Gateway to connect it to your network, start UPStart and connect to the Gateway. To do this, open the UPStart interface selection dialog, choose the Gateway as the interface type, discover its address and port, and then connect. For now, leave the username and password empty.

If you have not already used the normal features of UPStart to create a test now would be a good time to do that. Make sure you save the UPB file to your disk. Then export it into the Gateway. Press the “Export” button in the “PulseWorx Gateway” ribbon category. The necessary tables are written to the Gateway memory. Now close UPStart. You must do this or PulseWorx-App can’t connect since UPStart is a pulse-mode application and in that case no other connections are allowed as long as it is connected.

After installing the PulseWorx application, start it. Press the “Configure Connection” button in the “Gateway” ribbon panel to open the Interface setup dialog. In the interface configuration dialog choose the Gateway and test the connection with the “Test Connection” button. Close the interface selection dialog. From the ribbon “Home” category, press the “Connect” button. This connects to the gateway and if it asks if the state tables should be retrieved - you should do that.

During connection, the application reads the UPStart export file (.UPE file) from the Gateway and also reads the tables containing the state of the network - maintained by the Gateway as UPB messages are sent and received. If you just started the Gateway then the tables will be empty and everything will appear to be off. If you control some devices, then exit and restart the application you can see that the application now displays the state of devices in sync with the actual devices.

The user interface is very simple. An icon is shown for each room in the network. A tap (left mouse click or touch screen short tap) on a room icon shows a page with icons for each device. A tap on a device toggles its load. A tap on a keypad displays a popup keypad whose buttons show the state of the actual keypad LED. A tap on one of the keypad buttons executes the single-click action as programmed into the keypad button.

The ribbon “Home” category contains:

- Buttons for connecting to the Gateway and configuring the powerline interface. When enabled, for testing purposes, it can also work with the other PIM types.
- A button to get or set the Gateway clock.
- Schedules. The Gateway has up to four schedules with one schedule being the “current” schedule. The current schedule is the one that the Gateway watches for actions to perform at given times. Scheduling is accomplished in “Weekly” or “Calendar” mode. The “Schedule mode” button opens a dialog that explains these concepts and lets the user select which is used.

- A “Back” button. After opening a room to display the device in that room the “Back” button returns to the home page.

In addition, if developer mode is enabled – instructions below – a “Development” category is available in the ribbon. This contains:

- Open a viewer that shows all messages between the Gateway and the application. In a later section of this doc an example of what the viewer shows is given.
- Open a viewer that shows Gateway diagnostics messages. This is probably of no interest but may be needed to diagnose problems during the development period.
- A tool to set the location saved in the Gateway. The location is needed to construct tables that make it possible for the Gateway to have the correct sunrise and sunset time for schedule execution.
- A tool to set the Gateway clock
- A tool to define up to 4 authorized users and what actions they can perform
- Open a viewer that reads and decodes the device state table. This is one of the tables applications read to sync their state with the network
- Clears the device state table. This is done by writing a table with all entries set to unknown (0xff)
- Open a viewer that reads and decodes the link state table.
- Clears the link state table. This is done by writing a table with all entries set to unknown (0xff)
- Decode Activate Table, Decode Deactivate Table, Decode Device Type Table, Decode DST Table. These operations display a decoded version of the tables UPStart creates that allow the Gateway to maintain state.
- A tool to create the “reset table” that holds the reset password.
- File Tools. These actions allow access to the Gateway file system where the UPE file and all the tables are stored. Facilities are available to move files into and from the Gateway as well as list the Gateway file directory and delete files.
- PIM Modes: This is for testing and allows the application to communicate with the PIM in pulse or message mode. All applications except UPStart should use message mode.

Communicating with the Gateway

Before looking at the PulseWorx-App source, the first place to start is to review the “Gateway Firmware Design” document. It contains a description of all the facilities implemented in the Gateway. It describes the commands, and the protocol and message formats. Then to augment this documentation, you can review the PulseWorx-App sources to see how the Gateway facilities are used.

A key concept to understand when working with the Gateway is that at times you are talking to the Gateway firmware (to read and write files for example) and other times you are communicating with the PIM (to send and receive UPB messages, to read and write PIM setup registers). It is important to remember that all commands sent to and received from the Gateway are in the Gateway message format. Inside that message will be data. When talking to the PIM, that data is formatted as the PIM expects it.

If you are reworking an existing implementation that talked to a serial/USB PIM or the PIM-IP then all your commands that you send and receive are in PIM format. Those need to now be “wrapped” in Gateway protocol.

PulseWorx-App: Guide to the implementation

Here are the major tasks that any application working with the Gateway needs to perform and where to look in the sample application source to see an implementation of that task.

Gateway Discovery

The protocol for discovery of the Gateway is covered in the Gateway Firmware Design doc starting on page 2. It is implemented in these app sources:

- ISelect.cpp. Look at the method OnClickedDigiDiscovery
- GatewayDiscovery.cpp. The UDP messages are sent, received, and decoded in this class.

Gateway commands and protocol

The Gateway command protocol described in the Gateway doc at the end of page 4 and the start of page 5 is implemented in these sources:

- Gateway.h. This file defines constants for all Gateway commands, fixed tables name and table sizes, and error codes.
- UPBPIM.cpp. This is the main class for implementing communication with all variants of the PIM and as such is large file. For examples of Gateway packet construction a good example is:
 - SendGatewayClockRequest. In here you can see an example of packet construction.

- The function GatewayChecksum implements the checksum as described in the Gateway Firmware Design doc.
- For decoding of packets look at the function PacketCheckGateway. Note that during initial connection to the Gateway when the first messages are sent and received, the packet protocol is not used. The start of this function checks to see if it is using the connect mode protocol or the packet protocol

Gateway files and tables

There are a number of files in Gateway file system constructed by and read by applications. The names of these are defined in Gateway.h and if they have a fixed size then the size is also defined there.

The native file system created by the OS in the Digi-Connect uses case sensitive filenames. To rationalize this, the Gateway firmware lowercases all filenames passed in and out. The filenames are limited to the old DOS 6.3 convention.

While a number of these filenames are fixed – those that are used by the Gateway firmware – there are no restrictions about applications placing their own files in the gateway.

The file commands are described in the Gateway Firmware Design doc starting on page 8. These are used in GatewayFile.cpp. It uses facilities implemented in UPBPIM.cpp to send commands and receive responses. It also uses facilities in UPBPIMUtil.cpp to coordinate the sending of commands and waiting for responses – with error and timeout handling.

In MainFrm.cpp start at the function OnDirList to see how it uses facilities in UPBPIM.cpp to request a directory listing.

The tables and where they are read/written in PulseWorx-App are:

- devastate.dat: Holds the state of all devices in a UPB network. Its format and use is described in the state keeping section. Look in the file MainFrm.cpp, ConnectFileHandler function to see how the file is read and decoded to update the application state
- Inkstate.dat: Holds the state of each link (scene) in the network and tells if it was last activated or deactivated.
- Inkact.dat, Inkdact.dat, devtype.dat: Tables constructed by UPStart and provided a way for the Gateway firmware to quickly update its state tables when a link is activated. Its format is described later in this doc.
- users.dat: Implements the user table that defined authorized users with a name, password, and permission settings. If the user table exists then only users defined in it can successfully connect. The permissions control what files they can access and/or write to. The user table is defined in the table format section. Also look in the GatewayUserAccess.cpp file.

- `schedule.dat`: Holds the schedule entries that controls the autonomous scheduler in the Gateway. The format of this table is defined in the scheduling section. The data storage of the schedule is defined in the `GatewayScheduleEntry.cpp` file. A user interface for creation and editing of schedule entries is in `GatewayScheduleEntryEdit.cpp`.
- `calendar.dat`: How a 5-year calendar used by the Gateway when “calendar schedule mode” is used. The format and use is described in the scheduling section.
- `suntime.dat`: Holds the sunrise and sunset times for each day of the year. The table format is described in the scheduling section. Look at `GatewayUI.cpp` in the `GatewayUILocation` function since changing the location needs to change this table.
- `dst.dat`: Holds the start and end dates for daylight saving time for 10 years. The table format is described in the scheduling section. Also look in the `GatewayUILocation` function.
- `Location.dst`. This table is not used by the Gateway – all the Gateway firmware cares about is in the `suntime` and `dst` tables - but is used by PulseWorx-App to save the location selected by the user. Its format is described in the table format section.

Implementing connection authorization

The connection facilities described in the Gateway Firmware design doc starting on page 2 are implemented in two main locations. These are:

- `UPBPIMUtil.cpp`. Look at the `PulseWorxGatewayConnect` function.
- `MD5.cpp`. This is an implementation of the MD5 hash used during connection. Rather than use the Windows security facilities, the hash was implemented directly in the application. Most system runtimes on mobile platforms have some version of MD5 that perhaps can be used directly.

At the end of this document is a section on the Connection Protocol with an example.

Reading a UPE file

The UPE file – the UPStart export file is the key piece of information that an application needs to understand the UPB network. It provides in a “digested” form all the key device parameters, the configuration of transmitters and receivers. The documentation of the format is in the “UPStart Export File Description” pdf file.

PulseWorx-App reads the export file – named “`export.upe`” - decodes it and saves the data for each device in a `CUPBDevice` class - `UPBDevice.cpp`.

Also the class UPBNetwork (UPBNet.cpp) contains the overall network information read from the export file as well as all the link names.

Also contained in the UPBNetwork class is a reader for the export file itself – function UPBImport. Review the “UPStart export file” format document to follow along with the import.

The UPE file format has grown over time from a simpler format. Unfortunately the format is what it is at this point and it has several “quirks” that can make reading it challenging. The best way is to carefully follow the sample implementation in UPBNet.cpp and using that as a model for your own implementation.

Reading and extracting state information from the device and link state tables

The format of the device and link state tables is in the table format section. These tables are read and decoded during connection after the export.upe file has been read. Look in mainfrm.cpp, ConnectFileHandler method.

Keeping application device state up to date from UPB messages received

As described in the state keeping section the initial state of the network can be read by an application from the state tables maintained by the Gateway firmware. After an application is connected then the Gateway send update messages to keep connected application device state up to date.

Format and use of the User table

The user table format is documented in the table format section. A reader and writer of this table is in the file GatewayUserAccess.cpp. It is expected that few applications will read or write the user table. Probably only done by UPStart or similar UPB Network configuration programs.

Setting time and location

To set the clock in the Gateway the location is also needed. The Gateway firmware needs to know if the current day is in the Daylight-Saving Time range and also the offset from GMT. An implementation of setting the clock can be found in GatewayClock.cpp

Note: Currently, the offset from GMT is not used but the expectation is that at some point the Gateway will periodically connect to an internet time service to keep the clock accurate. In this case it needs to know the offset from GMT to use these services.

Reading and writing the schedule table

The format of the Schedule table is defined in the scheduling section. PulseWorx-App and UPStart implements a UI to display, create, and edit schedule entries. The key sources to look at are:

- GatewayScheduleEntry.cpp: The definition of schedule entry: It also contains facilities to encode and decode from the Gateway format into the class variables.
- GatewayScheduleDisplay.cpp and GatewayScheduleEntryEdit.cpp: A user interface for editing schedule entries.

Once the schedule table is assembled then the same file facilities in the Gateway are used to store the schedule.

As described below in the scheduling section, each schedule entry can be tagged with a number from 0 to 3. This lets a user divide up the 256 possible schedule entries into 4 possible groups. An application can direct the Gateway to consider only schedule entries tagged with a specific number. This gives the user the ability to have up to 4 schedules that they can be used for whatever purposes needed. The start of the schedule table contains the names for these schedules that a user can set.

An implementation of the command to switch between schedules is in GatewayCurrentSchedule.cpp

Handling pulse mode and message mode applications

As described in the Gateway Firmware Design doc only a single Pulse Mode application can be connected to the Gateway at a time. When an application requests to enter pulse mode then all other clients currently connected to the Gateway are disconnected.

The Gateway message 0xf0 is sent to all connected clients. This message is processed in PulseWorx-App in UPBPIM.cpp and results in a message sent to mainfrm.cpp and is processed in the OnDisconnectMessage function.

Sending and receiving UPB messages

As described in the “UPB Powerline Interface Module: PIM Description” doc, the PIM and application can receive commands in either pulse mode or command mode. Sending messages looks the same in either mode. In general, command mode is much simpler to work with and most applications will use it. The CUPBPIM class implements both.

To see an example of formatting a command to send to the PIM, look in PIM.cpp at a function like SendGoto. It is important to remember when talking to the Gateway that you first must assemble the UPB command in valid UPB packet, then surround it with the necessary characters that the PIM expects – to send a message on to the powerline the UPB packet is in text format, starting with ctrl-t (or ‘T’) and ending with a CR (see PIM description doc, page 8). Next that complete PIM message must be placed in a Gateway packet and then sent to the Gateway for onward shipment to the PIM.

Since the CUPBPIM class works with all PIM variants, that creating of the Gateway packet is held to the last and is implemented in the AddToSendQueue function.

To see an implementation of UPB command message decoding, look in the PIM.cpp function PacketCheckPIM_CommandMode.

One important note for all applications: UPB messages can be sent in single packets or in multi-packets. As described in the UPB Technology Description doc, these are expressed in the CNT and SEQ fields of the UPB message control word. This allows messages to be sent of the form 1 of n. This is the setting that is configured on a device by device basis called “Transmit Count” and is usually found on the “Advanced” tab of a device’s properties in UPStart.

One difficulty that happens with multi-packets is that it is important for a transmitter not to transmit over another transmission. This can become a problem in this scenario:

1. A message is sent to a device, for example to retrieve its status. It is sent in a 2-time packet (CNT = 01).
2. The device responds in a 2-time packet.
3. A message is sent to another device as soon as the 1 of 2 message is received. This can collide with the 2 of 2 message not yet received.

Suppose that an application was attempting to retrieve the status of several devices. It sends out the status request, receives the response in the 1 of 2 packet and since it receives the response, it then immediately sends out the status request to the next device. Unfortunately, this collides with the 2 of 2 response from the first device.

A poor implementation would know that a 2-time packet response is expected and just wait for the reception of the 1 of 2 and the 2 of 2 messages before proceeding. This, unfortunately, doesn’t work as the 2 of 2 transmission could be lost due to powerline noise.

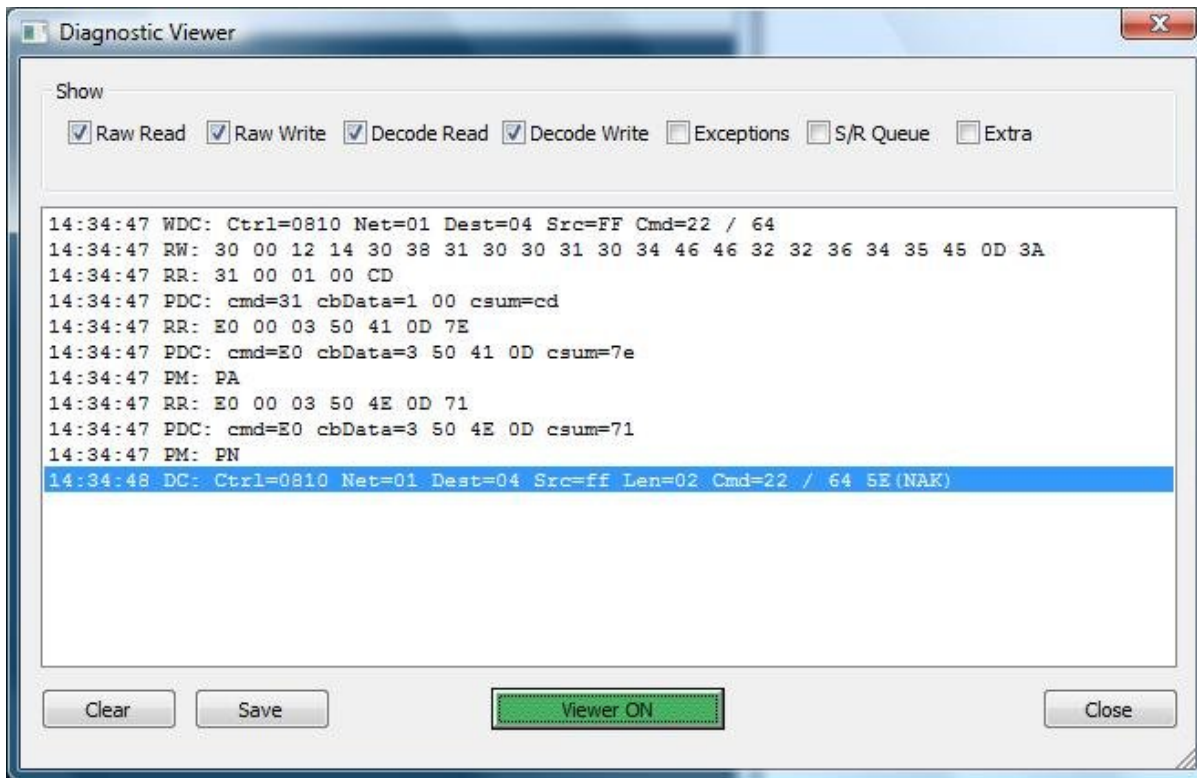
The only way to handle this correctly is to understand the timing of UPB messages and how long to wait after receiving the 1 of n message until sufficient time has elapsed to received all responses. This is implemented in the PIM.cpp as the “Blackout time” and you can see how that is computed and handled in NetworkSendTask when communicating with the Gateway.

Note: Unless you are sending commands in a stream – for example polling ‘n’ devices for status – you probably don’t need to concern yourself with this.

How to listen in on the Application to Gateway communications

The best method to learn the Gateway protocol – aside from this documentation – is to perform actions with the PulseWorx-App and watch the data sent to and received from the Gateway.

To do this, press the Comm Viewer button in the Development category. This opens a viewer.



Using the app a device was controlled to 100%. This is what the viewer shows:

Line 1, tagged “WDC”, is a Write Decode of the UPB packet to be sent.

Line 2, tagged “RW” is the Raw Write of the bytes sent to the Gateway.

Line 3, tagged “RR” is a Raw Read from the Gateway.

Line 4, tagged “PDC” is the Packet Decode of the Gateway message. In this case it is responding to say that the message was received and processed without error.

Line 5, tagged “RR” is again a Raw Read from the Gateway.

Line 6, tagged “PDC” is the Packet Decode of the Gateway message. In this case it is a message from the PIM.

Line 7, tagged “PM” is the PIM Message decoded. In this case it is “PA” which tells that the PIM accepted the message for transmission.

Line 8, tagged “RR” is the Raw Read of a message from the Gateway.

Line 9, tagged “PDC” is a decode of that message. In this case another message from the PIM

Line 10, tagged “PM” is the PIM message. In this case PN which means that the device didn’t ACK the command. This is ok. While having the ACK bit set in the message control word, this network being used for this test has a Split Phase Repeater (SPR) which renders the ACK bit meaningless.

Line 11, tagged “DC” is a Decode of the complete UPB message and the ACK/NAK.

As you can see this trace mechanism, while useful it can be very verbose. As such you can control the type of messages logged by using the checkboxes in the dialog.

Keeping current status of a UPB network

To maintain the status of a UPB network the state for each device in the network must be tracked. This is complicated by two things:

- Some devices have multiple channels. There are several two channel PulseWorx products. The largest number of changes is a Simply Automated Inc. 4-channel relay device.
- Keypads have button LEDs. The most LEDs on a keypad are eight. For convenience purposes we can consider these LEDs in the same way that channels are used for multi-channel devices

Status needs to be kept for keypad buttons if an application wishes to show “Glass Keypads”. A glass keypad is a physical representation of a keypad presented to the user so that they can “push” a button. Since UPB keypad buttons can be set to a toggle action (activate – deactivate – activate - etc) it is important that the button appears to the user in a manner that shows “on” or “off” so the user knows what the action of the button will be before they “press” it.

An aside on “glass keypads”: The reason for this UI paradigm is that users may not know or remember the underlying action of the keypad button. A button in general activates or deactivates a link so you might suppose that a user could just activate or deactivate that link to achieve the same action. The problem is that they may not know or remember that link. All they may know is that they want to press the “go to bed” button on the “hall keypad” and whatever results from that button press is the desired action. A Glass Keypad with labeled buttons achieves that.

The Gateway firmware watches the commands that are sent from the PIM and from applications sending commands into the PIM. It watches for activate, deactivate, goto, and status report commands. The status report command is sent by a device that may not be otherwise linked but is configured to report status on local activation.

In order for the firmware to keep state, UPStart builds and places several tables in Gateway file system.

NOTE: When an application connects to the Gateway it should read the device and link state tables to determine the state of the network. While connected it should process state update commands to maintain that state. Both are explained in the next sections.

The Device State Table

The Device State table keeps the state of all devices in the UPB network. The table is organized as 256 rows of 9 columns per row. (256 bytes * 9 bytes). Each row of the table provides the state for one device in the network. Each column represents the state of each device channel or keypad LED. The values are in the range 0-100 or 255 if the state is unknown.

For a simple single load device, only the 1st column of a row contains useful data.

For a keypad load dimmer/relay, the 1st column contains the state of the load and the next 8 columns are the state of each keypad LED (0 = off, 100 = on)

For a keypad that doesn't have a load, then the 1st column has no meaning. The next 8 columns contain the state of the button LEDs.

For an input device (ICM, TCM, DBM), column 0 contains the state of the 1st input, and column 1 contains the state of the 2nd input. When the input is closed the state is reported as 0, and when the input is open the state is reported as 100.

For the 2-output, 3-input I/O Module (IOM) - available from Simply Automated or Web Mountain and in wide use – column 0 and column 1 contains the state of the 2 outputs. Columns 2, 3, and 4 contain the state of the inputs. Like the ICM, an open input has a state of 100 and closed as 0.

The Link Activate and Deactivate tables

Note: These tables are used by the Gateway firmware and usually are not usually read or written by applications other than UPStart. They are presented here for completeness.

These tables are built by UPStart and are written as part of the "Gateway Export" action and are named LNKACT.dat and LNKDEACT.dat. Unlike other tables in the Gateway, these tables have no fixed size. Their size is determined by the number of devices and the number of links in use.

One table is indexed by the link number from an Activate or Link-GOTO commands and the other on the link number from a Deactivate or Link-GOTO command with level 0.

Ultimately what the table determines is the level that a device goes to as a result of receiving a link activate or deactivate.

The activate and deactivate tables are organized in two sections.

- The first section is a 256 entry array of 2-byte quantities indexed by link number. The array elements contain an offset from the start of the table to the entries for this link. If the offset is zero then there are no actions for this link.
- Link info at the offset: 2 bytes which contains the count of 3 byte quantities. This is followed by "count" 3-byte quantities as described below.

Each action is composed of three bytes:

- Module id
- Device channel number
- Level (0 to 100)

The module id is an index into the device state table and determines the table row. The channel number is used as an index into the Device State table and determines the column.

Processing a UPB message using these tables is simple. The MDID (the command) is extracted from a UPB message and if it is one of the commands that matter as listed above, then the link number is used to index into the table header and the offset is followed to get to the list of 3 byte actions which are then processed. Each action describes the device, channel, and level to be updated. Thus, these tables have pre-computed the action of each command based upon the contents of devices “receive components” table.

If the format of the table is unclear, look at the PulseWorx-App implementation of the “Decode Activate Table” and “Decode Deactivate Table” commands available in the “Development” ribbon category, “Table Tools” panel.

The State Update Command

When UPB messages are received by the Gateway it uses the above tables and processes the message to determine its effects – the “Device State Report” UPB message is also processed. As the effects of that UPB command are determined - as defined by these tables - “State Report” messages sent from the Gateway to all connected clients. The format of this command is defined in the Gateway Firmware document. This message – command code 0xE0 – always has ten bytes of data. The first byte is the UPB module id of the device that is affected and the remaining 9 bytes are used in the same way as a row in the state table.

Note: The Gateway sends status update messages based upon what devices are potentially affected. For example, if a device channel is at 100% and the command effect would have it go to 100% then a status update is still generated even though the level is unchanged.

Schedule actions in the PulseWorx Gateway

The PulseWorx Gateway firmware implements an autonomous scheduler that executes schedule entries from one of four schedules that send UPB activate, deactivate, or GOTO messages to specified scenes. In many ways, this scheduler is similar to the PulseWorx TEC Timed Event Controller product.

Applications can implement a user interface for letting users create and modify schedule entries and when complete store that schedule into the Gateway. For example, both UPStart and PulseWorx-App has a user interface for this.

NOTE: One of the permissions of a user as contained in the user table is the ability to modify the schedule table.

There are a few terms to get used to with scheduling:

- **Schedule Entry.** A schedule entry defines a specific action at a specific time. For example, to activate a link at 6pm.
- **Schedule.** A schedule contains one or more schedule entries.
- **Current Schedule.** While the Gateway has up to four schedules only one is active at a given time. The current schedule is either changed by the user manually – weekly scheduling – or by the Gateway everyday at midnight – Calendar scheduling.
- **Schedule Action.** Each schedule entry causes a link to be activated, deactivated, or GOTO. It is important to remember that this is a link (scene) based scheduling system. A user doesn't schedule a specific light to come on, they schedule a specific link to be activated and that may have the effect of controlling one or more lights.

The Gateway implements two different scheduling modes: Calendar and Weekly. Different installations may benefit from using one or the other.

In weekly scheduling you select the days of the week that each schedule entry applies to. This can be very efficient if the installation has scheduling needs that are oriented to the days of the week and not specific to calendar dates. For example, turning the lights on at 6am Monday to Friday every week. When using weekly scheduling you may have other schedules that are used for specific purposes – for example when away from home or on vacation. In weekly mode it is possible to have limited “date specific” schedule entries. They have to be manually entered with a month and day.

With calendar scheduling there are also up to 4 schedules. For each day of the year you specify which of the four schedules is in use for that day. Calendar scheduling is appropriate if the schedule is dependent upon the day of the year. For example, a schedule that reflects having a holiday or school vacation schedule different from the normal schedule. For calendar scheduling the Gateway has a 5-year calendar table that determines the schedule to use for each day. The format of this is specified below.

Determining the appropriate scheduling mode is determined by the answer to the question: Does this installation have scheduling needs on specific days of the year or just on the day of the week?

The table that drives the scheduler is named “schedule.dat” and is read, modified, and written by applications. The Gateway firmware processes the table each minute to see if schedule entries should be executed.

The schedule table starts with a single byte that is followed by four 16 byte names. These are the schedule names and are of no interest to the firmware but used by the application in presenting the schedules to the user.

The first byte of the schedule table contains:

- The high bit (0x80 mask) determines if calendar or weekly scheduling is in effect. If the bit is zero then weekly scheduling is used, if 1 then calendar scheduling is used.
- The low two bits (0x03 mask) determines the transmit count when the Gateway sends commands created when a schedule entry is executed. It is important to set this correctly! If the UPB network contains a repeater this must be set for 2, 3, or 4 time packets (encoded as 0x01, 0x02, 0x03).

The remainder of the table contains 64 possible events. Each schedule entry consists of these parts:

- Schedule id
- The link to be used (1-250)
- The action: Activate, Deactivate, GOTO 100%, GOTO 0%
- The date. This is represented by either a month-day for a specific date in the year or by a bit map with one bit for each day. In calendar scheduling this should be set to 0xff
- The time. These options are available:
 - A specific time (HH:MM)
 - At sunset
 - ‘n’ minutes before sunset
 - ‘n’ minutes after sunset
 - At sunrise
 - ‘n’ minutes before sunrise
 - ‘n’ minutes after sunrise
- A link that controls the schedule entry: If the link is activated then this schedule entry is suspended and if deactivated then this schedule entry is resumed. This lets an installation create, for example, a keypad button that prevents one or more scheduled actions from being

executed at their schedule time.

- A “vary” amount associated with each schedule entry. The idea is that an amount plus or minus the specified “vary” is applied to the computed or specified time in the entry before being carried out. This is to introduce some variability in the schedule. If not needed then the vary amount is set to 0.

In the table an event is represented by 11 bytes:

- [0]: Schedule id (0 – 3)
- [1]: Event link
- [2]: Event “suspend” link
- [3]: Time Type
 - 0 = HH:MM
 - 1 = Sunset plus delta
 - 2 = Sunset
 - 3 = Sunset minus delta
 - 4 = Sunrise plus delta
 - 5 = Sunrise
 - 6 = Sunrise minus delta
- [4]: Date Type
 - 0: MM-DD
 - 1: Weekday bitmap (In calendar scheduling this is always set to 1)
- [5]: Command
 - 0 = activate
 - 1 = deactivate
 - 2 = GOTO 100% (See the UPB description document for the differences between using Activate and Goto)
 - 3 = GOTO 0%
- [6]: Date1: For calendar dates, the month. For a weekday bitmap organized as xSMTWTFS. If the bit is set then the schedule entry applies on that weekday. For example, a value of 0x7f has the schedule entry apply on all weekdays. A value of 0x40 would have the schedule entry apply only on Sunday.
- [7]: Date 2: For calendar dates, the day. For all other date types it is unused.
- [8]: Time 1: For HH:MM times, the hour. For all other time types it is the sun-time delta as an unsigned number. The time type tells if the delta should be subtracted or added to the sun time.
- [9]: Time2: For HH:MM times, the minute. For all other time types not used.
- [10]: Vary: The vary amount given as an unsigned number

Any schedule entry that has a schedule id value not in the range 0-3 is ignored. Upon encountering the first table entry with a schedule is greater than 3, then the remainder of the table is considered empty.

To support the “sun time” relative times, an application that allows setting the location – UPStart does this – must build a table and store it in the Gateway named “suntime.dat”. The table contains (12 * 31) entries indexed by (month * 12 + day). This table organization makes it much simpler to find the data for a given date since all months have 31 days. Each table entry contains 4 bytes which are:

- Sunrise hour
- Sunrise minute
- Sunset hour
- Sunset minute

To support daylight-saving time, an application that allows setting the location – UPStart does this - builds a 10-year table that specifies when standard and daylight saving time begins and ends. This table – named DST.DAT” - contains 10 records each 5 byte long. The record bytes are:

- Year – 2000
- Month DST starts
- Day DST starts
- Month DST stops
- Day DST stops

If the location doesn’t support DST then the tables bytes are all 0xff.

The calendar.dat table is a table organized as:

- One byte of the current year – 2000.
- Eight bytes described below
- 5 * (12 * 31) entries indexed by (month * 12 + day). Each entry contains a number in the range 0-3. This is the number of the schedule to have as the current schedule on that day.

As you can see the calendar contains data for 5 years. The first 12 * 31 bytes are for the 1st year – the year specified by the first byte of calendar.dat, the next 12 * 31 bytes for the next year, etc.

In the unlikely event, the gateway is not reprogrammed for 5 years eventually the current year will “go beyond” the end of the calendar. In this unlikely scenario bytes 2 to 9 (the 8 bytes of the table referenced above) are used as they provide the schedule to use (0 – 3) of each day of the week. The first byte is for Sunday, the second byte for Monday, etc. The eighth byte is unused. In UPStart and PulseWorx-App the user selections in the Calendar setup tool are saved in these bytes.

NOTE: If it wasn't clear from the above, an application that supports weekly schedules reads and writes schedule.dat. An application that supports calendar scheduling reads and writes schedule.dat and calendar.dat. Only applications that allow setting of the location need write dst.dat and suntime.dat.

PulseWorx Gateway Table Reference

The Gateway contains a file system for storage of a number of files (tables) that control its actions. The tables can be grouped into several categories:

- System tables
 - The user name/password table (users.dat)
 - The network information table (network.dat)
- Tables written that describe the UPB network and allow the Gateway firmware to keep track of network state. Generally these are written only by applications that modify the UPB network rather than control it (UPStart and UPStart like programs)
 - Link activate and link deactivate (lnkact.dat and lnkdact.dat)
 - Device type (devtype.dat)
- Tables written to control the scheduler
 - The schedule entries table (schedule.dat)
 - The sunrise and sunset table for the location (suntime.dat)
 - Daylight Saving time change date tables (dst.dat)
 - Calendar table (calendar.dat)
- State tables that are read by all applications that access the Gateway and want to sync their state to the network.
 - Device state (devstate.dat)
 - Link state (lnkstate.dat)

In addition to these tables, the UPStart export file (export.upe) is also stored in the Gateway. This allows applications to understand the network.

Also stored is a file called location.dat that is written by applications to save the location entered by the user that determines the values in the sun time tables. This file is used only by applications and not by the Gateway firmware.

The Gateway file system is open to applications storing other files in the file system. The only difference between these tables and any tables placed by the application is that these specifically named tables have their access checked against the file access permissions for the current logged in user.

The Gateway firmware implements a permission system for access to the various tables that reside in the Gateway file system. Because of this, the application does not need to implement its own permission system but rather can rely upon the Gateway to determine which tables the user can read and which the user can write. In this way it determines the permissible actions of the user. All the application need do is to display the appropriate error to the user if a permission violation occurs.

The Gateway Firmware Design doc describes how this permission system works in greater detail.

What follows is a brief description of each table and its format.

DEVSTATE.DAT

The Device State table keeps the state of all devices in the UPB network. The table is a 256 by 9 byte table. Each row of the table provides the state for one device in the network. Each column represents the state of each device channel or keypad LED. The values are 0-100 or 255 if the state is unknown.

For a simple single load device, only the 1st column of a row contains useful data.

For a keypad load dimmer/relay, the 1st column contains the state of the load and the next 8 columns are the state of each keypad LED (0 = off, 100 = on)

For a keypad that doesn't have a load, and then the 1st column has no meaning. The next 8 columns contain the state of the button LEDs.

For an input device (ICM, TCM, DBM), column 0 contains the state of the 1st input, and column 1 contains the state of the 2nd input.

This table is $256 * 9 \text{ bytes} = 2304 \text{ bytes}$

This table can be read by any user and written by users with write table permission.

LNKSTATE.DAT

The link state table keeps track for each link if the link was last activated or deactivated. It is a 256 byte table with one row representing one link. If the link was last activated the array entry contains 100, otherwise 0.

This table can be read by any user and written by users with write table permission.

LNKACT.DAT and LNKDEACT.DAT

The Link Activate and Deactivate tables contain directions for how to process link activate and deactivate commands and how devices are affected by a link. These tables are built by UPStart and saved into the gateway file system when a design is exported to it. These tables have no fixed size.

The activate and deactivate tables are organized as:

- The first part of the table is a 256 entry array of 2-byte quantities indexed by link number. Each array element contains an offset from the start of the table to the entries for this link. If the offset is zero then there are no actions for this link.
- Link info at the offset: 2 bytes which contains the count of 3 byte quantities. This is followed by “count” 3-byte quantities as described below.

Each action is composed of three bytes:

- Module id
- Device channel number
- Level (0 to 100)

The channel number is used as an index into the Device State table row for a device. Use of these tables is more fully described in the State Keeping section.

This table can be read by any user and written by users with write table permission.

DEVTYPE.DAT

To assist in processing status device status report messages the device type table gives an indication of the type of device that sent the message.

This table is organized as is a 256 byte array indexed by the module id of each device in the network.

The possible values in the device type array are:

- 0 = normal load
- 1 = Keypad
- 2 = keypad with load
- 3 = Input-Output module
- Another other value = ignore

This table can be read by any user and written by users with write table permission.

USERS.DAT

The users table contains the user name, password, and privileges of users who can login to the Gateway. The table is organized as 4 rows. Each row contains:

- 17 bytes user name
- 17 bytes password
- 1 byte bitmap privileges

Applications that write this table are expected to zero terminate the user name and password. Because of this, the applications should limit the user name and password to 16 characters thus allowing the zero terminator byte.

The 1 byte bitmap contains these bits:

- 0x01: If set the user can write the schedule table
- 0x02: If set the user can write all tables (doesn't apply to the USER or NETWORK table)
- 0x04: If set the user can write the network and users table

If the user table is not present then all connections are accepted and no authentication is asked for when a client connects to the Gateway.

This table can be read or written to only by users with user/network table permission.

NETWORK.DAT

The network configuration file is used by the Gateway to determine upon startup how it should configure its network access. If the file is not found then the Gateway is configured with DHCP enabled and so obtains its IP address from the DHCP server. The table is 23 bytes long and organized as:

- 1 byte (0x01 = DHCP if set, Static IP if that bit not set)
- 4 bytes Static IP address (ignored if using DHCP)
- 4 bytes Subnet mask (ignored if using DHCP)
- 4 bytes Gateway (ignored if using DHCP)
- 4 bytes DNS
- 4 bytes Secondary DNS
- 2 bytes TCP Port (default = 2101)
- 6 bytes MAC address. Initially stored when the Gateway is programmed at PCS

The multi-byte quantities are stored in with the most significant byte preceding the least significant byte. For example a port number of 2101 (hex 0x0835) is encoded as two bytes as 0x08 0x35. This is different than the multi-byte storage of numbers on X86 platforms.

SCHEDULE.DAT

The schedule table is used by the Gateway firmware to implement its autonomous scheduler. The table contains a header of 65 bytes organized as a single byte followed by four 16 byte schedule names. These names are expected to be zero terminated so applications should limit the schedule names to 15 characters.

The first byte of the schedule table contains:

- The high bit (0x80 mask) determines if calendar or weekly scheduling is in effect. If the bit is zero then weekly scheduling is used, if 1 then calendar scheduling is used.
- The low two bits (0x03 mask) determines the transmit count when the Gateway sends commands created when a schedule entry is executed. It is important to set this correctly! If the UPB network contains a repeater this must be set to 2, 3, or 4 time packets (encoded as 0x01, 0x02, 0x03).

The remainder of the table contains 256 schedule entries. Each is represented by 11 bytes:

- [0]: Schedule id (0 – 3)
- [1]: Event link
- [2]: Event “suspend” link
- [3]: Time Type
 - 0 = HH:MM
 - 1 = Sunset plus delta
 - 2 = Sunset
 - 3 = Sunset minus delta
 - 4 = Sunrise plus delta
 - 5 = Sunrise
 - 6 = Sunrise minus delta
- [4]: Date Type
 - 0: MM-DD
 - 1: Weekday bitmap
- [5]: Command
 - 0 = activate
 - 1 = deactivate
 - 2 = GOTO 100%
 - 3 = GOTO 0%
- [6]: Date1: For calendar dates, the month. For a weekday bitmap organized as xSMTWTFS. If the bit is set then the schedule entry applies on that weekday.
- [7]: Date 2: For calendar dates, the day. For all other date type it is unused.
- [8]: Time 1: For HH:MM times, the hour. For all other time types it is the sun time delta as an unsigned number. The time type tells if the delta should be subtracted or added to the sun time.
- [9]: Time2: For HH:MM times, the minute. For all other time types not used.
- [10]: Vary: The vary amount given as an unsigned number

This table can be read by any user and written by users with write table or write schedule table permission.

CALENDAR.DAT

This table contains the calendar used by the scheduler when in calendar scheduling mode.

- One byte of the current year – 2000.

- Eight bytes described below
- $5 * (12 * 31)$ entries indexed by $(\text{month} * 12 + \text{day})$. Each entry contains a number in the range 0-3. This is the number of the schedule to have as the current schedule on that day.

The eight bytes are the default schedules to use if the current date read by the Gateway clock is past the end of the table. The first byte of the eight is the schedule to use on Mondays, the second byte of the eight is the schedule to use on Tuesdays, etc.

SUNTIME.DAT

The table contains $(12 * 31)$ entries indexed by $(\text{month} * 12 + \text{day})$. This table organization makes it much simpler to find the data for a given date since all months have 31 days. Each table entry contains 4 bytes which are:

- Sunrise hour
- Sunrise minute
- Sunset hour
- Sunset minute

This table can be read by any user and written by users with write table or write schedule table permission.

DST.DAT

To support daylight-saving time, UPStart a 10 year table that specifies when standard and daylight saving time begins and ends. This table contains 10 records each 5 byte long. A record bytes are:

- Year – 2000
- Month DST starts
- Day DST starts
- Month DST stops
- Day DST stops

If the location doesn't support DST then the tables bytes are all 0xff.

This table can be read by any user and written by users with write table or write schedule table permission.

LOCATION.DAT

This file stores the location used as set by the user. The location is used during construction of the sun time and daylight saving time tables. It is stored as a direct memory copy from this C structure:

typedef struct

```
{
// From the file
char cityName[32];
char stateName[32];
double latitude;
double longitude;
int UTCDeltaMinutesStandard;
BOOL dstObserved;

int zipcode; // If 99999 then the location was manually configured

// Daylight is standard plus one hour
int UTCDeltaMinutesDaylight;

int dstStartWeek;           // 1, 2, 3, 4, or 5
int dstStartWeekday;       // Sunday(0) - Saturday(6)
int dstStartMonth;         // Jan = 1

int dstStopWeek;
int dstStopWeekday;
int dstStopMonth;

// Number minutes plus/minus to correct computed sunrise/set to better
// match installation due to terrain. Set by the user
int localSunriseDelta;
int localSunsetDelta;
} LocationInfo;
```

Since this file is not processed by the Gateway, the byte ordering of the fields are in X86 format.

EVENTS.DAT

While the name of this table is documented in gateway.h, it is not used currently.

Additional Notes on Gateway connection

As described above and in the Gateway Firmware Design doc the initial connection to the Gateway is a process of sending and receiving messages. This example may provide some additional help.

To start a connection this is what PulseWorx-App sends. All this is text. The <0> terminates a read or write.

The connection example below is for the Gateway with a username and a password. If the password table is empty or doesn't exist than the Gateway sends back "AUTH NOT NEEDED" in the response to your message and the connection is complete.

There are also several possible errors that are in the response to the first message sent by the application. These are – in these exact text strings.

- MAX CONNECTIONS REACHED
- PULSE MODE ACTIVE
- PIM NOT INITIALIZED
- FIRMWARE CORRUPT – FLASH WITH UPSTART

It is important that all these get reported to the user in some form.

(Below, RW stands for "Raw Write", RR stands for "Raw read")

This is what the client sends to the Gateway after the socket is open.

12:25:21 RW: 50 75 6C 73 65 57 6F 72 78 20 41 70 70 2F 31 2E 30 2E 31 2F 31 00

This decodes to "PulseWorx App/1.0.1/1<0>" and is the name of the connecting application, the application version major dot minor dot build, and the protocol wanted. Protocol "1" is all that is implemented currently. The Gateway doesn't do anything at this time with the application name or version information.

This is what comes back from the Gateway

12:25:21 RR: 50 43 53 20 50 49 4D 2D 49 50 32 2F 31 2E 30 2F 31 2F 41 55 54 48 20 52 45 51 55 49 52 45 44 2F 34 39 32 36 44 31 30 35 32 31 30 35 30 35 46 43 37 31 33 33 37 42 39 31 30 30 44 46 42 41 37 43 33 30 44 45 32 45 34 33 32 38 39 41 36 42 42 41 34 38 44 35 45 44 39 43 32 35 41 33 33 33 31 45 33 39 31 34 46 41 46 35 34 31 37 36 34 42 36 42 31 44 45 31 38 36 30 32 37 46 33 37 33 42 46 38 34 32 30 30 34 41 43 41 30 37 42 39 30 45 37 31 34 31 46 31 33 37 36 45 34 33 46 39 41 39 37 43 00

Here is the decode

12:25:21 PCS PIM-IP2/1.0/1/AUTH
REQUIRED/4926D105210505FC71337B9100DFBA7C30DE2E43289A6BBA48D5ED9C25A3331E3914FAF5
41764B6B1DE186027F373BF842004ACA07B90E7141F1376E43F9A97C

The “PCS PIM-IP2” is a stock phase it always sends. The “1.0” is the gateway firmware version, the “1” is the protocol and then the phase “AUTH REQUIRED” followed by a bunch of stuff which is the challenge key.

To process this, convert the challenge key from 128 characters to 64 bytes. Then hash it with the password. The “Gateway Firmware Design” doc talks about how this is done. If you have the PulsWorxApp sources, look in the file “UPBPIMUtil.cpp” and the function “PulseWorxGatewayConnect” and also the MD5.cpp source.

The next message to the Gateway is the username, a slash, and then the 16 bytes from the hash rendered into 32 characters. Again, terminates by a zero byte.

12:25:21 RW: 6B 69 6D 62 65 72 6C 79 2F 38 45 33 43 39 39 37 42 44 44 39 30 37 45 37 41 30 43 43 39
44 44 38 37 37 33 42 41 41 42 45 44 00

This decodes to:

kimberly/8E3C997BDD907E7A0CC9DD8

If all goes well a message comes back a message from the Gateway:

12:25:21 RR: 41 55 54 48 20 53 55 43 43 45 45 44 45 44 2F 30 20 43 4C 49 45 4E 54 53 00

This decodes to:

12:25:21 AUTH SUCCEEDED/0 CLIENTS

And the connection is made.

From this point on all messages between the application and the Gateway are in Gateway packet format.

##end##



19201 Parthenia Street, Suite J
Northridge, CA 91234

P: 818.701.9831
F: 818.701.1506

pcssales@pcslighting.com
www.pcslighting.com