Using the PIM from your own application to send and receive UPB messages

It isn’t difficult to create applications that use the PIM to send and receive messages. The PIM interface and the UPB command set is fully documented. This app note brings together in one place information from several documents and provides a step by step “what you need to know” about using the PIM.

Note: This is an advanced topic and is only needed by users who are interfacing their own applications to the PIM.

Proper preparation prevents poor performance
The first step is to make sure that you have downloaded the complete documentation about the PIM and UPB. These documents can be found on the PCS web site under the following titles in the resources section.

- Powerline Interface Module PIM Description
- UPB Technology Description

It also would be a major help to have a most recent copy of UPStart and PulseWorx-EZ installed on your computer as they have tools that are explained here and will be useful.

Note: In this note I will refer to the Powerline Interface Module as a “PIM”. Everything in this note applies to it or the serial model “CIM” from Simple Automated. Simply Automated also has a USB version of the CIM product that is communicated with using a USB connection and not using the usual serial port commands described here.

Note: In this note anytime I want to refer to a byte value I will do so in hex prefixed with “0x”. The byte value 1 will be written as 0x01, the value 2 as 0x02, etc.

Basic need to know
All communication with the PIM, either the serial version with the true serial port or the PulseWorx USB PIM that through a device driver creates a virtual serial port, uses basic serial communication. A serial port to the PIM must be opened at 4800 baud with the usual N-8-1 settings (No parity, 8 bit bytes, and 1 stop bit).

Communication with the PIM is possible in two different modes: Message mode and Pulse mode. You can read about the differences in the PIM description document. The bottom line is that you want to use message mode and there is no advantage in using pulse mode for any application except UPStart.

The PIM should power on in message mode. If it doesn’t appear to – some application left it in pulse mode, you can always get it back to message mode by using the button on it and tapping 5 times, then the LED will start to blink. Tap 10 times – like resetting a PulseWorx device – and then after the LED changes color, 2 more times to return to normal operation.
All messages sent and received, expect for the characters that head and tail a message, are in printable text. While the text represents hex numbers the messages are sent as text. To clarify: If you wanted to send the byte value 0x5F you would send two bytes of data. The first contains a 0x35 and the second a 0x46. These are the ascii codes for a “5” and a “F”.

There are two kinds of messages you can send to the PIM. The first type is a message that includes the UPB command to send on to the power line. The second type of message is to read or write data from/to the PIMs own memory. You do not need to use the commands for reading and writing the PIM internal memory to send and receive UPB messages. Read about them if you want in the PIM description document.

**Basic PIM Communication**

When sending messages to the PIM the general protocol you should follow is:

1. Send the message to the PIM
2. Wait for a response from the PIM
3. Process the response which could cause you to send the message again or not
4. Possibly process the response from the device

The idea here is that your application shouldn’t “fire and forget”. The reply to each message should be checked and handled.

For each message sent to the PIM there are four possible responses that the PIM sends back:

- Accept
- Busy
- Error
- No response

“Accept” means that the PIM has accepted your message and will then process it. “Busy” means that the PIM is unable to accept your request. You should wait a tiny amount of time and send the message to the PIM again. “Error” means that the PIM found some problem with your message - generally the message is incorrectly formed. You could resend the message to the PIM – there could have been a communication glitch – but you probably don’t want to do that more than once as communication failures are rare. If you get an error the 2nd time, then your application is almost certainly wrong in composing the message.

The final response of “no response” is a recognition that any communication protocol that is implemented will handle all possible errors. There should be no reason the PIM shouldn’t not reply to your message but your application should handle it perhaps by resending the message.

**Format of PIM Messages**

Each message sent to the PIM begins with a control character and ends with a <cr> character – hex 0x0d.
When sending a message to the PIM instructing it to send a message out onto the power line, the character starting the message is a `<Ctrl-T>` (a 0x14). If you have a more recent generation PIM you can use the character ‘T’ (a 0x54) instead.

The character that terminates the message is a `<cr>` (a 0x0d). If you have a more recent generation PIM you can use the character ‘Z’ (a 0x5a) instead.

In summary, to send a message to the PIM it looks like:

```
<ctrl-t>UPB-MESSAGE<cr>
```

The next section covers with that “UPB-MESSAGE” looks like.

As described above, when the PIM receives that message it replies back with:

- PA<cr>
- PB<cr>
- PE<cr>

These are the PIM messages for “Message accepted” (PA), “PIM is busy” (PB), “The message contains an error” (PE). As described above you should check for these messages and handle them as appropriate.

**UPB messages**

Now we turn to what a UPB message looks like – its component parts – and how to compose messages that perform certain actions – like activating a scene or controlling a device ON or OFF.

**UPB Message control word**

Each UPB message contains these parts:

- Two bytes that make up the “control word”.
- The destination network id
- The destination unit id
- The source unit id
- The command
- Arguments to the command if any
- Checksum

The control word is the most complex part of the message. In the UPB Technology Description document there is a good picture of it on page 12. Here are the parts you need to know.
Link Bit
The one bit (called “Lnk” in the UPB doc) that says if this message is for a device or scene. Each message can be sent to control an individual device – on, off, set a level – or control a scene – activate or deactivate. This bit in the control word determines if the message is controlling a device or a scene.

Message Length
The 5 bits that make up the message length (called “Len” in the UPB doc). Each message contains several bytes and this provides the number of bytes in the message. This length includes the two bytes of the control word, the destination network id, the destination unit id, the source unit id, the command, any command arguments and the checksum. The smallest useful message would be 7 bytes (one byte for a command with no arguments) and the largest message is 24 bytes.

Note: Don’t get confused when reading the UPB Description doc. In there it talks about a “Preamble Byte”. Ignore that as it isn’t relevant to communication at this level.

Message Acknowledgement
The 3 bits that determine if/how a device responds to receipt of a message (called “ackrq” in the UPB doc). When a UPB device receives a command, it can signal that the message was received. There are two ways that can be done: An “ack pulse” and an “ack message”.

The “Ack pulse” is a quick way that the device signals it received the message and the PIM has direct support for it – more on this later. The “Ack message” is a complete UPB message that the device sends back as an acknowledgement and to process it your application would have to both send UPB messages and also receive and process them. One limitation you should be aware of is that the “Ack pulse” can’t be relied upon when a UPB installation has a SPR or TPR.

If you are sending to a single device, then the ACK bit or ACK message can be useful. If you are sending a scene command then many devices can potentially respond to the scene so the ACK message or ACK pulse isn’t useful as you can’t know easily which did respond.

Send Count
The 2 bits of the message count field (called “cnt” in the UPB doc) determines how many times the message is sent onto the power line. This isn’t a dumb method where the message is just repeated exactly each time. The SEQ field (2 bits) tells the receiver if this message is 1 of 1, 1 of 2, 2 of 2, 1 of 3, etc. In that way, a receiver can see how good or poor the communication is. Is it receiving all the multiple messages or only some? UPB devices act upon the first good message it receives – usually 1 of n – and then ignores the other transmissions of the message.
Also in the message control, word is a field called “REPRQ”. This isn’t useful in almost all cases and should be set to 0.

**Network Id**

To find the id of a network, open the file in UPStart and on the “Network” ribbon category click on the “Properties” button.

The network id is given as a decimal number and you will need to convert it to hex for use. In this example, the network id is 136 decimal or 0x88.

**Destination Id**

The next field in the UPB message is the destination Id. If you are sending a message to control a scene (the link bit in the control word is a 1, then this is the scene id. If you open a UPB file with UPStart you can find the id number assigned to each scene. In the “Scenes” ribbon category click on the “Builder” button:
For example, the “Library On” scene is assigned scene id 7. Remember that the numbers seen in UPStart are decimal! You must remember to convert them to hex for us. So when UPStart shows a scene id as 21, when constructing messages the destination id will be 0x15.

To find the id of a device, load the file into UPStart and open the properties of the device.

<table>
<thead>
<tr>
<th>ID Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unit ID</td>
</tr>
<tr>
<td>Network ID</td>
</tr>
<tr>
<td>Manufacturer ID</td>
</tr>
<tr>
<td>Product ID</td>
</tr>
<tr>
<td>Firmware Version</td>
</tr>
</tbody>
</table>

The unit id is displayed as a decimal number and you will need to convert it to hex for us. In this example, the unit id is 24 decimal or 0x22.

**Source Id**
The source id can be whatever you want but the convention is to use 0xff when an application is using the PIM to send commands.

**Command and Arguments**
The next field is the UPB command id (called a MDID in the UPB doc). There are many of these and the ones that you probably will use come from the “Device Control Command Set”. These are documented in the UPB Description doc on page 40.

These are the commands that cause devices to do things: Go on, off, to a level, blink, etc. Each has a single byte command id and some have additional arguments.

Rather than repeating all that the UPB Description doc says this is the time to refer you to that other documentation as it tells what the command is and the arguments are in all the detail you need.

**Checksum**
The checksum byte contains a value that the PIM uses to determine if it received a message from you with all the bytes received by it the same as those that you sent. In short, this tries to detect communication failures.

The checksum is easy to compute: Sum all the bytes of the message starting with the 1st byte of the control word, and ending at the last argument to the command, then take the 2’s compliment of the sum (or multiply by -1) and truncate the result to 8 bits.
Sending the Message
Now that all the fields of the message have been assembled, head the message with a <ctrl-t> and terminate with a <cr> and send it to the PIM.

ACK / NAK
After the PIM receives the message it replies with a PA, PB, or PE. Assuming you get the PA then the UPB message is sent on to the power line.

After the message is sent the PIM next sends to you a PK or PN message. This tells you that the message was sent and the power line is now free to send another message. But it also reports if the device acknowledged the command or not.

If you have the bit in the message control word that request that the destination responds with an “Ack pulse”, and if the device did transmit that ack pulse upon receiving the command, then the PIM sends the PK message. If you didn’t request an Ack pulse or you did and the device didn’t acknowledge, then the PIM sends to you the PN message.

Receiving Messages
All long as you have a serial port open to the PIM, the PIM send messages to your application when UPB messages are seen on the power line. The messages are prefixed with a PU the sent to your application in the same “hex text” as always. The message from the PIM terminates with a <cr>.

Trying this all out with some tools
There are three tools you should make use of – two in UPStart and one in PulseWorx-EZ – that may help you with better understanding communication with the PIM and UPB messages.

UPStart contains the UPB Explorer and the Command Wizard
The UPB Explorer is a way you can encode and decode UPB messages. Open it from a button on the “Tools” ribbon category.

Note: For some older versions of UPStart, to get to the UPB Explorer you must first open the UPStart options dialog from the application menu, then on the “Extra” tab enter the code UPB1. Then the UPB Explorer button is available on an “Advanced” Ribbon category.
The UPB explorer allows you to construct any UPB message setting any of the fields in the control word. Use the checkboxes to set the control word fields, enter the message source and destination - network id, destination id, and source id. Then enter in the “Send” box the command and any command arguments and press the Send button. The UPB Explorer set the Length and Checksum for you as you enter the various settings but prior to sending you can override those to see what happens on errors.

Don’t forget that all the ids and commands are all in hex. Too many times I have seen device id 14 in UPStart and used that in UPB Explorer -0 and not 0x0e as I should have – and then struggled to figure out what is wrong.

In the display sends are marked with a triangle pointing to the right and received with a triangle pointing left. If you select a line in the display, at the bottom the message is decoded to show you the various message components.
UPB Command Wizard
In the “Tools” ribbon category there is also the “Command Wizard” tool. This is a multi-step wizard where you choose all the elements of a UPB message and the wizard produces the exact text to send to the PIM to carry out that command.

The wizard takes you through choosing a network id, destination id – scene or device control, source id, all the various fields in the control word, device command and command arguments. Then the completed command displays:
PulseWorx-EZ PIM Comm Viewer
While the UPB command Wizard and UPB Explorer in UPStart are very good at working with UPB messages at the lowest level, to see the PIM interactions between the application and the PIM – the PA, PB, PE, and PU commands, PulseWorx-EZ – available for download from the PCS web site – may be useful as well.
Once you have PulseWorx-EX running with a PIM connected and a file loaded, select from the application menu “Power line interface” – “PIM Comm Viewer”.

Once you click on that button a viewer opens and it can be left open while you control devices in your design. Click on the button at the bottom of the viewer – changes from red to green to show it is active – and then it starts showing all the messages sent between the PIM and the application. In this example I right-clicked on a device and selected ON from the popup menu.
The first line shows the components of the message about to be sent, the 2\textsuperscript{nd} line shows the actual bytes written, the 3\textsuperscript{rd} line shows the PIM responding that all is well, and the 4\textsuperscript{th} line reporting that the device didn’t ack. The last line is a recap of what was sent.

Using this tool you can see how PulseWorx-EZ constructs UPB messages and how it interacts with the PIM.

\textbf{Advanced Ideas}

In this app note only the control commands were focused upon. But the UPB documentation has all the possible commands. You can write applications that uses the PIM to program devices like UPStart, request device state, enumerate devices to find all the devices in a network and more. That is where the UPB documentation helps with all the details.

There are other application notes for communicating with the PIM-IP and the PulseWorx-Gateway. But while you send and receive messages with a different communication method than the serial PIM, the composition of UPB messages as described in this note is the same.

###end###