ONE
NET

wireless control for everyone

ONE-NET PEER MESSAGING

Version 2.3.0

August 14, 2012

Table of Contents

1 Overview **4**

2 "One Time" Versus "Semi-Permanent" Pairing Of Device / Unit Pairs **4**

2.1 Example Of A "One Time" Message *4*

2.2 Example Of A Semi-Permanent Device / Unit Pairing Using The ONE-NET Peer Table *5*

2.2.1 Peer Table Contents 5

2.2.2 Adding A Device / Unit Pair To The Peer Table 6

2.2.3 Removing A Device / Unit Pair From The Peer Table 8

2.2.4 Forwarding A Status Change To The Peer Table 10

2.2.5 Overall Process Starting From When The Light Switch 0 Is Flipped On 11

3 Other Setups Using Peer Messaging **13**

4 Repeater And Source DIDs When Using Tree Or Linked List / Message Forwarding **17**

1 Overview

Peer to Peer Communication in ONE-NET is not a rigidly defined concept. Network designers can define it however they like. Generally, though, it means semi-permanently tying the status of logical units on different devices to each other. That's not a very satisfactory definition, but read on!

2 "One Time" Versus "Semi-Permanent" Pairing Of Device / Unit Pairs

Any device can send a message to any other unit of any other device in the network. This can be done as a "one-time" message or the two devices / units can be officially paired together as a ONE-NET "peer table" entry.

2.1 Example Of A "One Time" Message

Suppose, for whatever reason, there are two physical devices, each with a transceiver. Device 002 has 2 logical units, numbered 0 and 1. Both are light switches. Device 003 has 4 logical units, numbered 0 through 3. All logical units on device 003 are lights.

Suppose that switch 0 on device 002is flipped on. It can send out a status change message to unit 2 of device 003. Device 003 sees the status change message, and being a light, assumes that it is supposed to turn unit 2 on. The interpretation of a status change message is application-dependent, but in general, for devices like relays, it is assumed that a status change message should be treated as a command.

It's completely unclear why unit 0 of device 002 decided to tell unit 2 of device 003 its new status. For whatever reason, the application code of device 002 decided to do that. It may do that every time. It may not. It did it without any input from the master. It did so without the help of the Master device.

2.2 Example Of A Semi-Permanent Device / Unit Pairing Using The ONE-NET Peer Table

Imagine the same situation, only this time, device 02 uses the ONE-NET peer table mechanism.

Peer communications using the peer table are always set up by the Master device. The goal, generally, is for the master to set up the peer table, then get out of the way. This reduces complexity, increases speed, and lowers message traffic. One way to have unit 2 of device 003 paired with unit 0 of device 002 is to have device 002 send the master a message saying that unit 0 has changed status, then have the master send a message to unit 2 of device 003 telling it to change its status. This requires multiple messages and is particularly slow if devices 002 and 003 are not in range of the master without using multi-hop. This is unnecessarily complicated. It also ties up the master's resources for no good reason. A better solution is to use a peer table so that devices 002 and 003 communicate directly without the master needing tom intervene.

We assume that somewhere, at some time, a human being decided what switches should control what lights. That decision may or may not change. Hence the phrase "semi-permanent" might actually be "permanent" depending on the system. Regardless, at some point, a human being has to press a button on the master or type a command informing the master that it wants light switch 0 of unit 002 to control light 2 of unit 003. When that occurs, the master will inform device 002 to add an entry to its peer table via an ON_ASSIGN_PEER ONE-NET administration message.

2.2.1 Peer Table Contents

Assume the capacity of the peer table is 5. The peer table starts out empty. Each entry of the peer table consists of three items…

1. Encoded Device ID of The Peer Device
2. Source Unit
3. Peer Unit

ONE_NET_DEV_UNIT has the value 0x0F and denotes an empty element in the table. Similarly, 0xB4B4 decodes to 0x000 and represents an empty element in the table. Hence if an entry has 0xB4B4 as the Device ID or 0x0F as one of the units, it is considered to represent the end of the table. The table starts out empty.

Device ID	Source Unit	Peer Unit
0xB4B4	0x0F	0x0F
0xB4B4	0x0F	0x0F
0xB4B4	0x0F	0x0F
0xB4B4	0x0F	0x0F
0xB4B4	0x0F	0x0F

Figure 2.1 – Empty Peer Table. Blue Entries Are Empty

2.2.2 Adding A Device / Unit Pair To The Peer Table

A device implementing a peer table will maintain a table of a pre-determined capacity determined at compile-time by the ONE_NET_MAX_PEER_UNIT (please see the **ONE-NET Configuration Options, Features, And Port Constants Guide** document for more information on this constant). In addition, another value will be defined called ONE_NET_MAX_PEER_PER_TXN (please see the **ONE-NET Configuration Options, Features, And Port Constants Guide** document for more information on this constant). For this example, assume that the value of ONE_NET_MAX_PEER_UNIT is 5 and the value of ONE_NET_MAX_PEER_PER_TXN is 3. Upon receiving the “Assign Peer” message from the master, device 002 needs to do three things…

1. Check whether Device 003, Unit 2 is already in its table. If so, no changes are made and the message is ACK’d.
2. Verify that it actually has a unit numbered 0. If not, it will NACK the message.
3. Verify that it has room in its table to add the new entry.
    a. If the table is full, the entry cannot be added and the message will be NACK’d.
    b. If the table is not full, but there are already 3 entries with a source unit of 0 (recall that ONE_NET_MAX_PEER_PER_TXN is 3), the request will also be NACK’d.

c. If there is room on the table, the entry will be added and the message will be ACK'd.

In this case, since device 002's peer table is empty, there is room, so it will add the new pairing to the table. Note that the ENCODED device ID is stored. 0x003 encodes to 0xB4BA. Hence the entry {0xB4BA, 0, 2} will be added to the table, which ties unit 0 to unit 2 of Device 003. The new table is as follows.

Device ID	Source Unit	Peer Unit
0xB4BA	0x00	0x02
0xB4B4	0x0F	0x0F
0xB4B4	0x0F	0x0F
0xB4B4	0x0F	0x0F
0xB4B4	0x0F	0x0F

Figure 2.2 – Updated Peer Table. Blue Entries Are Empty

Suppose that three more entries are added. Unit 0 should also be paired with unit 3 of device 003, unit 1 should control unit 5 of device 004, and unit 1 should also control unit 2 of device 005. The encoding of device 0x004 is 0xB4B5. The encoding of device 0x005 is 0xB4B9. Hence the updated table will look like this.

Device ID	Source Unit	Peer Unit
0xB4BA	0x00	0x02
0xB4BA	0x00	0x03
0xB4B5	0x01	0x05
0xB4B9	0x01	0x02
0xB4B4	0x0F	0x0F

Figure 2.3 – Updated Peer Table. Blue Entries Are Empty

2.2.3 Removing A Device / Unit Pair From The Peer Table

Similarly, a peer assignment can be removed from the peer table with the ON_UNASSIGN_PEER command. Suppose that unit 1 of device 002 no longer controls unit 5 of device 004. Again, recall that 0x004 encodes to 0xB4B5. The master sends an ON_UNASSIGN_PEER message to device 002 telling it to remove 0xB4B5 / Unit 5 from the peer table. Device 002 receives this message and searches the peer table. If the entry is not in the table, then nothing is changed and the message is ACK'd. If it is in the table, as it is in this case, it will be removed and the entries below will all move up to fill in the gap. The last entry will be made empty to denote the end of the table.

Device ID	**Source Unit**	**Peer Unit**
0xB4BA	0x00	0x02
0xB4BA	0x00	0x03
0xB4B5	0x01	0x05
0xB4B9	0x01	0x02
0xB4B4	0x0F	0x0F

Figure 2.4 – Removing {0xB4B5, 1, 5} entry. Original List. Blue Entries are empty.

Device ID	**Source Unit**	**Peer Unit**
0xB4BA	0x00	0x02
0xB4BA	0x00	0x03
0xB4B5	**0x01**	**0x05**
0xB4B9	0x01	0x02
0xB4B4	0x0F	0x0F

Figure 2.5 – Removing {0xB4B5, 1, 5} entry. Entry found. Blue Entries are empty. Green entry is the relevant entry.

Device ID	Source Unit	Peer Unit
0xB4BA	0x00	0x02
0xB4BA	0x00	0x03
0xB4B9	0x01	0x02
0xB4B4	0x0F	0x0F
0xB4B4	0x0F	0x0F

Figure 2.6 – Removing {0xB4B5, 1, 5} entry. Green entry is removed. All entries below are moved up. Blue Entries are empty.

Device ID	Source Unit	Peer Unit
0xB4BA	0x00	0x02
0xB4BA	0x00	0x03
0xB4B9	0x01	0x02
0xB4B4	0x0F	0x0F
0xB4B4	0x0F	0x0F

Figure 2.7 – Updated peer table after unassign peer command. Blue Entries are empty.

2.2.4 Forwarding A Status Change To The Peer Table

Now we have an updated peer table. Switch 0 of Device 2 is flipped on. Device 2's peer table is below. For ease of reading, the raw device IDs are in parentheses next to the encoded Device IDs.

Device ID	Source Unit	Peer Unit
0xB4BA (003)	0x00	0x02
0xB4BA (003)	0x00	0x03
0xB4B9 (005)	0x01	0x02
0xB4B4	0x0F	0x0F
0xB4B4	0x0F	0x0F

Figure 2.8 – Device 002's Peer Table. Blue Entries are empty.

Unit 0 has had a status change with the flip of the switch. Its new status is 1 (representing "on"). When the switch flips, device 002 searches through its peer list and looks for any entries with a source unit matching the unit whose status just changed. In this case, that is unit 0.

Device ID	Source Unit	Peer Unit
0xB4BA (003)	**0x00**	**0x02**
0xB4BA (003)	**0x00**	**0x03**
0xB4B9 (005)	0x01	0x02
0xB4B4	0x0F	0x0F
0xB4B4	0x0F	0x0F

Figure 2.9 – Device 002's Peer Table. Entries where the source unit is 0 are green. Blue Entries are empty.

At the very least, device 002 needs to send its new status ("on") to units 2 and 3 of device 003. Device 003 should interpret those status change messages as commands and turn lights 2 and 3 on. In addition, it may need to send its new status to the master as well. To determine whether it needs to do so, it checks its "Send To Master" flag (see section 6 of the ONE-NET Specification document for more information on a device’s flags / settings). In this case, let’s assume that the master expects to be informed.

2.2.5 Overall Process Starting From When The Light Switch 0 Is Flipped On

Below is the expected overall process involved when a light switch is flipped on.

1. The application code of device 002 is either constantly polling the status of all switches or the switches are tied to an interrupt. Either way, the application code of device 002 determines that a status change has occurred on switch 0 and the new status is 1 (on).
2. Device 002 creates a “Status Change” application message and pushes that message onto the outgoing message queue, specifying that the message should be sent to the peer list.
3. The message is “popped” from the queue.
4. Upon popping the message, a recipient list must be created for the message. ONE-NET notes that the source unit is unit 0 and that the message should be sent to the peer list.
5. The peer list is searched for all entries where the source unit is 0. Two entries are found. Both entries are added to the recipient list. The recipient list now contains the following entries…
    a. Device 003, unit 2
    b. Device 003, unit 3
6. After adding all relevant entries from the peer list, ONE-NET notes that this is a status change message and that the device is a client device.
7. Therefore the “Send To Master” must be checked to see if the master device must be added to the recipient list.
8. The “Send To Master” flag is true, so the master must be informed.
9. The recipient list is searched to see if any of the recipients are the master.
10. No recipients on the list is the master, so another entry is added to the recipient list. {001, ONE_NET_DEV_UNIT}. Note, ONE_NET_DEV_UNIT stands for the device as a whole.
11. The recipient list is now as follows.

    a. Device 003, unit 2
    b. Device 003, unit 3
    c. Device 001 (master), unit F (Device as a whole)

12. The recipient list has been created. The recipient list is passed to the application code, which can adjust the recipient list if desired or cancel the message entirely. Let's assume that the application code approves and makes no changes. This message will have three recipients.
13. Are there any recipients left that have not been sent to?
14. Yes. Device 003, Unit 2.
15. A message is created and sent where the destination is Device 003, Unit 2.
16. Device 003 receives the message and turns light 2 on.
17. Device 003 ACKs the message.
18. Repeat step 13. Are there any recipients left that have not been sent to?
19. Yes. Device 003, Unit 3.
20. A message is created and sent where the destination is Device 003, Unit 3.
21. Device 003 receives the message and turns light 3 on.
22. Device 003 ACKs the message.
23. Repeat step 13. Are there any recipients left that have not been sent to?
24. Yes. Device 001, Unit F.
25. A message is created and sent where the destination is Device 001, Unit F.
26. Device 001 (the master) receives the message and does whatever it does with it.
27. Device 001 ACKs the message.
28. Repeat step 13. Are there any recipients left that have not been sent to?
29. No.
30. The entire transaction is complete.

At any time, the peer table can be changed. In this way, it is extremely easy to change what switches control which lights simply by having the master change device 002's peer table.

3 Other Setups Using Peer Messaging

In addition, a tree type or linked list structure can be set up where lights will forward messages to other lights. This allows the light switches to be able to control more lights, yet maintain smaller peer tables and spread the workload around more.

An example of this would be a single light switch controlling nine lights. Let's assume the light switch is device 002 and the nine lights are devices 003 through 00B. Let's assume further that each device has exactly one unit.

One method of having device 002 control all nine lights is to have all 9 entries in device 002's (the light switch) peer table. Another way would be to have 002 control lights 003, 006, and 009. In turn, 003 will pass the message to 004 and 005, 006 will pass the message to 007 and 008, and 009 will pass the message to 00A and 00B.

The corresponding tree and peer tables are below. Please note that, for easier readability, the peer tables are displayed using the Raw Device IDs. They are actually stored in memory as the Encoded Device IDs.

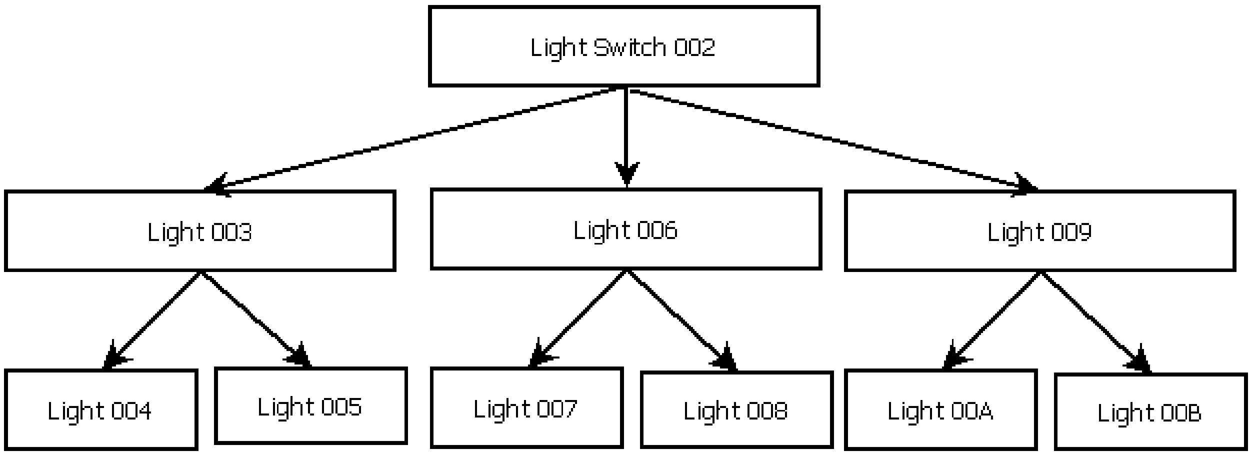

Figure 3.1 – Tree Representing Peer Setup Where Device 002 controls lists 003 to 00B In A Tree Setup.

Device ID	Source Unit	Peer Unit
0x003	0x00	0x00
0x006	0x00	0x00
0x009	0x00	0x00
0x000	0x0F	0x0F
0x000	0x0F	0x0F

Figure 3.2 – Device 002's Peer Table. Blue Entries Are Empty

Device ID	Source Unit	Peer Unit
0x004	0x00	0x00
0x005	0x00	0x00
0x000	0x0F	0x0F
0x000	0x0F	0x0F
0x000	0x0F	0x0F

Figure 3.3 – Device 003's Peer Table. Blue Entries Are Empty

Device ID	Source Unit	Peer Unit
0x007	0x00	0x00
0x008	0x00	0x00
0x000	0x0F	0x0F
0x000	0x0F	0x0F
0x000	0x0F	0x0F

Figure 3.4 – Device 006's Peer Table. Blue Entries Are Empty

Device ID	**Source Unit**	**Peer Unit**
0x00A	0x00	0x00
0x00B	0x00	0x00
0x000	0x0F	0x0F
0x000	0x0F	0x0F
0x000	0x0F	0x0F

Figure 3.5 – Device 009's Peer Table. Blue Entries Are Empty

In the scenario above, 002 sends the switch status to 003, 006, and 009. When they receive the messages, they will switch their lights on. In turn they will also search their peer tables and inform the lights on them, and those devices will turn on their lights. 003 will notify 004 and 005, 006 will notify 007 and 008, and 009 will notify 00A and 00B. Note how the peer tables relate to the graph. You can design as many branches of the tree as desired. This scheme can be extremely useful and allow a light switch to control lights outside of its range WITHOUT using multi-hop.

Similarly, one can easily imagine a linear progression of lights turning each other on… Assume that light switch 002 controls light 003, which controls light 004, which controls light 005.

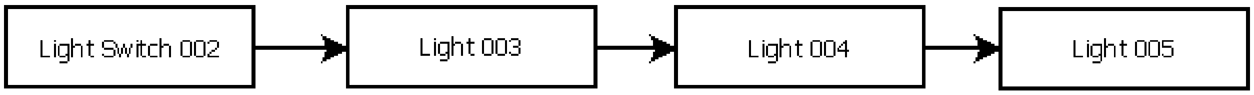

Figure 3.6 – Device 002 Controls Devices 003, 004, and 005 in a linear fashion

Device ID	Source Unit	Peer Unit
0x003	0x00	0x00
0x000	0x0F	0x0F
0x000	0x0F	0x0F
0x000	0x0F	0x0F
0x000	0x0F	0x0F

Figure 3.7 – Device 002's Peer Table. 002 is peered with 003. Blue Entries Are Empty

Device ID	Source Unit	Peer Unit
0x004	0x00	0x00
0x000	0x0F	0x0F
0x000	0x0F	0x0F
0x000	0x0F	0x0F
0x000	0x0F	0x0F

Figure 3.8 – Device 003's Peer Table. 003 is peered with 004. Blue Entries Are Empty

Device ID	Source Unit	Peer Unit
0x005	0x00	0x00
0x000	0x0F	0x0F
0x000	0x0F	0x0F
0x000	0x0F	0x0F
0x000	0x0F	0x0F

Figure 3.9 – Device 004's Peer Table. 004 is peered with 005. Blue Entries Are Empty

The end result is that 002 turns 003 on, which in turn turns 004 on, which in turn turns 005 on. The end result is a chain reaction with the status of all lights matching the status of the light switch.

4 Repeater And Source DIDs When Using Tree Or Linked List / Message Forwarding

When using a scheme such as the one above to forward messages, it’s generally best to have the Source DID be the light switch throughout the message propagation. The devices forwarding the messages should place their own Device IDs as the Repeater DID. This adds more information to the message since it lets all lights know that Device 002 is the device that initiated the message chain.