Powering Business Worldwide

White Paper

Modbus to INCOM Pass-Through Messaging Procedure

1. Introduction

As documented in Eaton protocol guides[1], the INCOM protocol supports reading and writing various configuration settings via the INCOM communications port.

For example, when communicating with a typical IQ/INCOM device such as the Eaton ATC-800 controller, the choice of which of the two sources is to be considered "preferred" can be selected via communications.

This is accomplished by

1. Reading (downloading) the setpoint buffer from the ATC.
2. Locating the portion of the setpoint buffer that selects preferred source
3. Changing that value to select either *Source 1*, *Source 2* or *None*[2] as the preferred source.
4. Recalculating the setpoint buffer checksum
5. Writing (uploading) the modified setpoint buffer (with new checksum) back to the ATC

While the Eaton protocol guides only describe the method for modifying the Eaton device setpoint buffer when using the INCOM protocol, this white paper describes a method that uses the Modbus protocol.

The Eaton INCOM protocol can be converted to and from Modbus using either of these two devices:

1. Modbus Product Operated Network Interface (Modbus PONI or MPONI for short)
2. Modbus Master INCOM Network Translator (Modbus MINT or MMINT for short)

Both of these devices permit a "pass-through" mode where INCOM messages can be imbedded as a payload within a Modbus message. While either the MMINT or the MPONI can be used, Eaton recommends using the MMINT rather than the MPONI as the MMINT supports larger data buffer transfers[3].

Special Modbus registers are reserved within the MMINT/MPONI for this INCOM pass-through method. When the MMINT/MPONI receives a message from a Modbus master directed to those registers, the data are extracted from the Modbus payload, converted to an INCOM message and transmitted to the Eaton device using the INCOM protocol.

---

[1] IMPACC Protocol Guides
http://www.eaton.com/ecm/groups/public/%40pub/%40electrical/documents/content/1030709222496.pdf (INCOM overview) and
http://www.eaton.com/ecm/groups/public/%40pub/%40electrical/documents/content/il17384f.pdf (ATS specific)

[2] "None" means that if the load is connected to a good source, the ATC will not attempt to move it to another.

[3] When selected for 9600 bps, the MPONI limits a Modbus message to no more than 61 registers. When operating at 1200 bps, the MPONI limits a Modbus message to no more than 10 registers at a time. The MMINT has neither limitation. You can still use the MPONI, you will just need to split read requests of more than 61 registers into two or more separate read messages.

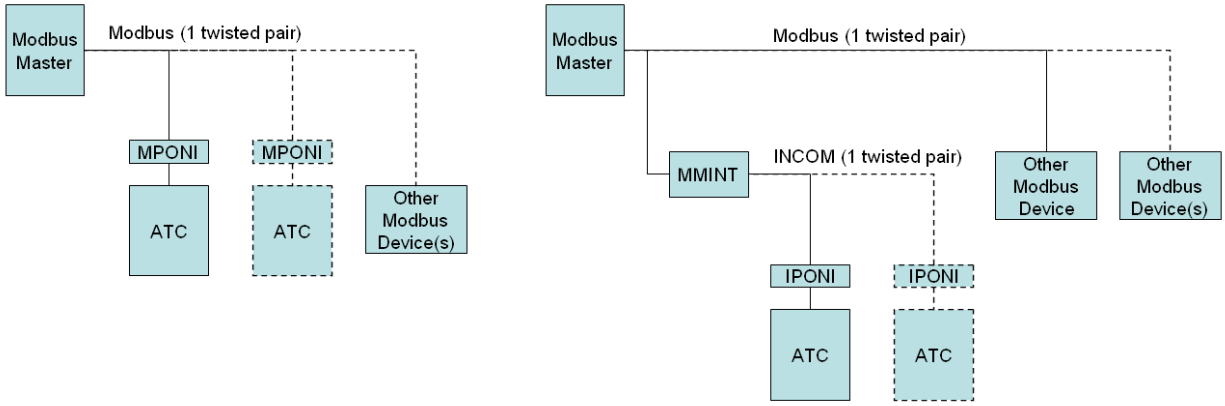

Figure 1: Left – MPONI wiring diagram, Right – MMINT with IPONI wiring diagram

As of the date of this document, the other Eaton devices that support Modbus protocol translation to and from INCOM (i.e. Power Xpert Gateway PXG-400 and the PXG-600) do not support the INCOM pass-through method and so cannot be used with this technique.

2. INCOM Pass-Through

Both the MPONI and the MMINT support pre-configured tables that are pre-populated with data from the connected INCOM device (or devices in the case of the MMINT). Data in these tables pre-populates once the MPONI or MMINT powers up and communications is established with the connected INCOM device(s). However if the data required by the Modbus master from the IQ/INCOM device is not pre-populated in the MPONI or MMINT tables, the user has the option of using something called INCOM Pass-Through.

This pass-through method allows a Modbus master to execute *any* available INCOM message supported by the INCOM device.

This pass-through method can, for example, be used to read or write a device's setpoint buffer (a set of values not pre-populated within the MMINT/MPONI).

By reading and changing the setpoint buffer, control commands or other actions within the INCOM device can be made. An example of this would be the ability to change the preferred source selection within an Eaton ATC-800[1] automatic transfer switch controller to a different value. The setpoint values are not available in the pre-populated tables within either the MPONI or the MMINT.

To use this method, however, requires more Modbus messages than just simply issuing a request for a block of data from the ATC.

The INCOM pass-through method requires that the user send a request to the MMINT/MPONI, wait for it to execute the command, then send another request to the MMINT/MPONI to confirm that the action took place successfully, and optionally retrieve any requested data.

[1] More information on this product can be found at http://www.eaton.com/ecm/groups/public/@pub/@electrical/documents/content/td15a09te.pdf. More information regarding the native INCOM communication protocol supported by this product can be found at http://www.eaton.com/ecm/groups/public/@pub/@electrical/documents/content/il17384c.pdf

Steps:

1. The user's Modbus master writes (using the Modbus preset register function code 16 dec or 10 hex) a message to the MPONI/MMINT that contains the embedded INCOM message intended for a particular INCOM device. Note that the Modbus slave address always matches the IQ/INCOM address, however, depending on whether the MMINT or MPONI are used, this address matching mechanism differs.

   Referring to Figure 1 on page 2, notice that the MPONI is connected one-to-one with an IQ/INCOM device. For the MPONI, the Modbus slave address and the INCOM address are the same value, and are set using rotary switches on the MPONI.

   For the MMINT the physical wiring layout is different. Notice that one MMINT can support multiple IQ/INCOM devices. With the MMINT, throughput alone establishes the only numeric limit. The twisted pair physical media itself supports up to 1000 IQ/INCOM slave nodes on the same twisted pair. However, since the MMINT can support multiple devices, the method of addressing the slave IQ/INCOM devices is for the MMINT to pass-through the IQ/INCOM address (as set on rotary switches on IQ device or within software of devices with embedded INCOM support). In other words, if an IQ/INCOM device has the address 10 hex, the Modbus address of that node will be 10 hex (16 decimal).

2. The next step is for the MPONI/MMINT to extract the INCOM payload from the received Modbus message and transmit it to the IQ/INCOM network (MMINT) or device (MPONI).

3. Depending on the type of message sent to the IQ/INCOM device, the IQ/INCOM device responds with either an acknowledgement (as would be typical for a Modbus write message) or with returned data (typical for a read message). The MPONI/MMINT stores the response from the controller in a separate Modbus register map within the MPONI/MMINT.

4. In a separate message, the Modbus master reads (using Modbus read registers function 03) block of Modbus registers to retrieve the response to the previous message. The data contained in these registers will vary depending on whether the INCOM message transmitted was a request for data or if it was a command that wrote data to the ATC.

   a. Request for data: Modbus registers will contain the data requested by the INCOM message

   b. Command to write data: Modbus registers will contain the acknowledgement of whether the message was received properly or if some error occurred.

Refer to Figure 2 for a diagram that explains this multi-step process in more detail.

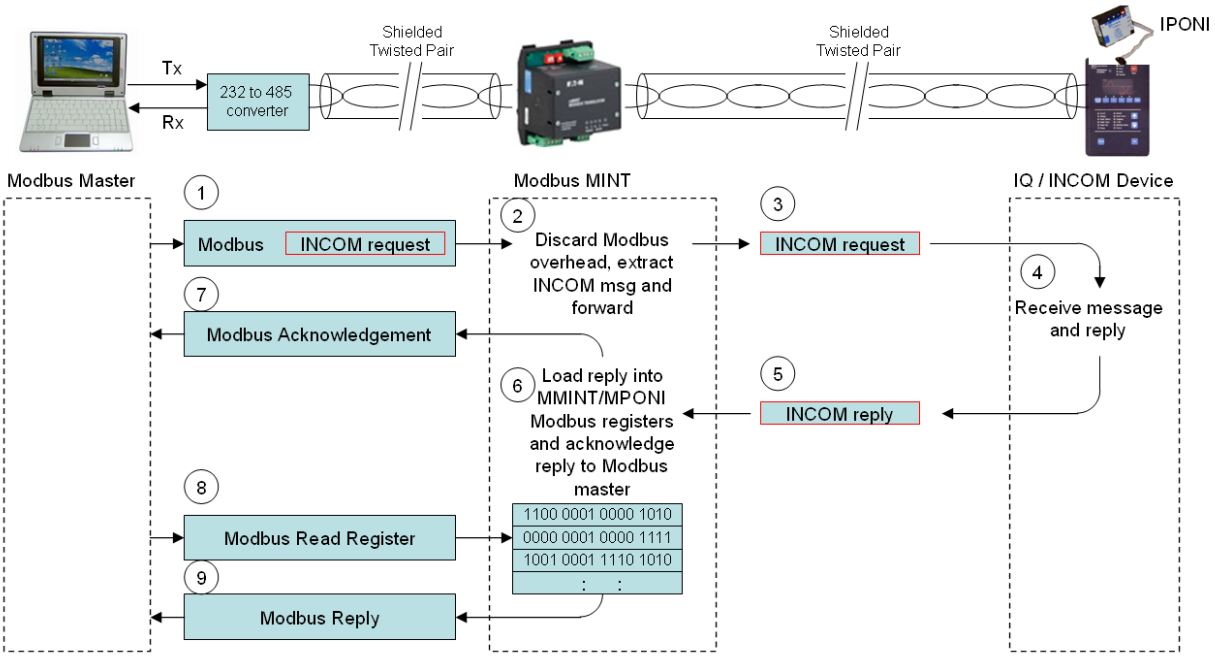

Figure 2 – *Two* Modbus messages are required for each Read INCOM Pass-Through

While reading data from an INCOM device is relatively straight-forward (two Modbus messages required as shown in Figure 2), the process for writing data *from* a Modbus master *to* an INCOM device using the INCOM pass-through method is surprisingly complex.

Figure 2 above and the step listed below explain in more detail how the INCOM pass-through method works with the MMINT or MPONI.

Steps

1 Modbus Master transmits INCOM request embedded within a Modbus Write Register (16 decimal or 10 hexadecimal) to the MMINT/MPOINI

2 MMINT/MPONI discards the Modbus overhead and extracts the INCOM payload.

3 MMINT/MPONI extracts the INCOM payload and transmit to the IQ/INCOM device

4 IQ/INCOM device receives, processes and replies to the message

5 IQ/INCOM device transmits reply (or replies) to MMINT/MPONI

6 MMINT/MPONI stores this data in a Modbus accessible register map

7 MMINT/MPONI transmits a Modbus reply that informs the Modbus master that the message was transmitted correctly. Note: the reply does not contain the data requested – only an acknowledgement that the message was received correctly by the MMINT/MPONI and that the IQ/INCOM device processed the message correctly. If the IQ/INCOM device rejected the message (wrong register, wrong number of registers, or some other problem), the Modbus reply will contain an error code.

8 A second Modbus message, this one a Modbus read (03) command must be sent from the Modbus master to the MPONI/MMINT to retrieve the data received from the previous Modbus message.

9 MMINT/MPONI responds with a reply containing the data requested.

The Modbus protocol[1] supports methods for reading and writing memory within Modbus slave nodes. In this document we will describe a Visual Basic 2010 program that uses these two function codes:

1. Modbus Function Code 3 – Read Holding Registers
2. Modbus Function Code 16 (10 Hex) – Preset Multiple Registers

While the maximum was not tested, the MMINT was confirmed to support Modbus read requests of the complete setpoint buffer (43 24-bit messages which corresponds to 86 Modbus 16-bit registers) in one message.

However, the Eaton MMINT/MPONI limits the number of registers that can be written during a single Modbus write command to only one 24-bit INCOM message. This corresponds to two 16-bit Modbus at a time. Writing a setpoint buffer with 43 messages would require 43 separate Modbus write messages. This is the primary reason for the increased complexity when writing data to an INCOM device using this method.

3. Setpoint Buffer Read/Write

As discussed in "Introduction", one example of using this technique would be to read and write a setpoint buffer in an automatic transfer switch controller and change the preferred source selection. As described in Eaton Instruction Leaflet IL 17384, Part F, Section 450[2], the (3 C 9) "Transmit Setpoints Buffer command returns to the requestor, all of the user selectable (and many of the factory preset) setpoint values. For the ATC-600 and -800 controllers, included in this list is the choice of which of the two sources is considered "preferred".

In particular, Byte 2 of Message 27 represents the low byte and Byte 0 of Message 28 represents the high byte of a 16 bit integer that can contain a number that is either 0, 1 or 2

- 0: No source is preferred. If connected to a valid source, no attempt will be made to switch
- 1: Source 1 is preferred. If Source 1 is good, an attempt will be made to switch to Source 1
- 2: Source 2 is preferred. If Source 2 is good, an attempt will be made to switch to Source 2

To change this source selection value, once this setpoint buffer is downloaded, the low byte of Message 27 can be changed[3], a new buffer checksum must be computed and stored as the final three bytes (Message 43, Byte 0, Byte 1 and Byte 2).

3.1. Checksum Calculation

In addition to the Modbus CRC included over the entire message, Message 43 of the data payload received as part of the response to the "Transmit Setpoints Buffer" command contains a checksum of the previous 42 INCOM messages contained in that payload.

This checksum increases the certainty that the data was received without corruption. Each of those 42 messages is contains a 24-bit value encoded as three 8-bit bytes. The checksum algorithm sums each of those 8-bit bytes separately without concern for location.

It is important for any program that wishes to transmit changed setpoints back to the ATC to be able to calculate this checksum accurately. A setpoint buffer transmitted to the ATC with an invalid 43rd message checksum will be rejected by the ATC.

---

[1] http://www.modbustools.com/PI_MBUS_300.pdf

[2] http://www.eaton.com/ecm/groups/public/%40pub/%40electrical/documents/content/il17384f.pdf

[3] The high byte will always be zero so no change is needed.

Recall from the INCOM protocol guide[1] that all INCOM data messages contain 24 bits, broken into three eight bit bytes (or octets) labeled [B2] for the most significant 8-bits, then [B1] for the middle 8-bits and then [B0] for the least significant 8-bits.

Since Modbus transmits 16-bits at a time, to send 24-bits, we need to send two 16-bit messages or 32-bits total.

Therefore when retrieving a large table of registers from an IQ/INCOM device using the MMINT/MPONI each 24-bit INCOM message will be stored in two 16-bit registers. Since two 16-bit registers contains 32-bits, the eight extra bits are allocated as "status" bits and provide information about whether the message was received correctly. Consult the appropriate MMINT[2] or MPONI[3] manual for more information on these status bits.

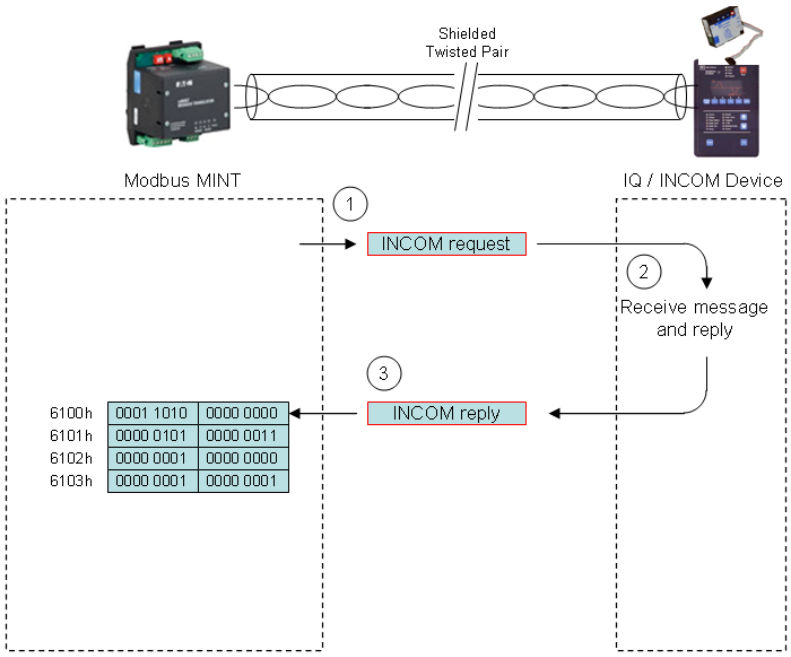

Figure 3

As an example, here is a partial listing of the data returned in response to a "Transmit Setpoints Buffer" command (an INCOM "3 C 9" command):

[1] http://www.eaton.com/ecm/groups/public/@pub/@electrical/documents/content/1030709222496.pdf

[2] http://www.eaton.com/ecm/groups/public/@pub/@electrical/documents/content/66a7508.pdf

[3] http://www.eaton.com/ecm/groups/public/@pub/@electrical/documents/content/66a2070.pdf

Table 1

Message	Hex			Raw (Base 16)	Raw (Base 10)	
1	6100	B0	H	2A	42	# addl msgs (42d, 2Ah)
		S	L	00	0	status
	6101	B2	H	05	5	FW version
		B1	L	03	3	FW revision
						Factory Set Options
2	6102	B0	H	01	1	0. TDES enabled
		S	L	00	0	status
	6103	B2	H	01	1	2. TDEN enabled
		B1	L	01	1	1. TDNE enabled
3	6104	B0	H	01	1	3. TDEC enabled
		S	L	00	0	status
	6105	B2	H	01	1	5. S2 OF monitoring enabled
		B1	L	01	1	4. S2 UF monitoring enabled
4	6106	B0	H	01	1	6. S2 OV monitoring enabled

This table shows the relationship between the INCOM message, the Modbus register address (shown as hexadecimal addressing), the raw data contained in the INCOM message and a description of that data. The first column “Message” is the INCOM message number. Each message contains 24-data bits (3 bytes) plus a status byte for a total 4 bytes per message.

The second column “Hex” shows the offset (in hexadecimal) from the beginning of the Modbus holding register table within the MPONI or MMINT (same address in either).

The third column (unmarked) has “B0”, “S”, “B2” and “B1”. These codes correspond to the position of the byte within the 24-bit INCOM message. In other words, a 24-bit value would be written as [B2][B1][B0], but would stored in the Modbus registers somewhat out of order as shown below.

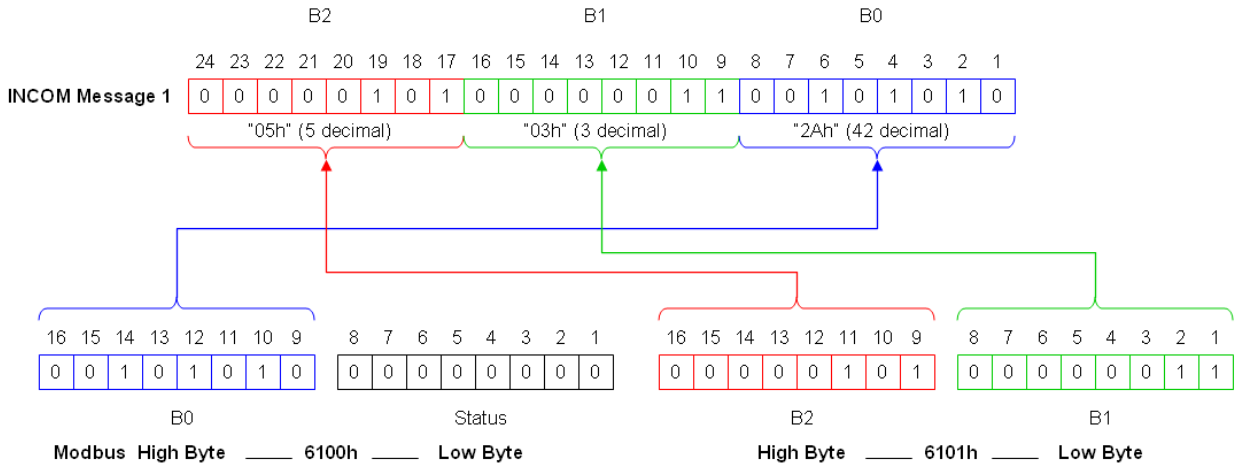

Figure 4

The status byte is transmitted with each message and should be used to determine if the MPONI/MMINT to INCOM device communication proceeded normally.

If the Modbus master had transmitted a request to retrieve two 16-bit Modbus register values from the MPONI / MMINT, that data would be returned in two INCOM messages as shown below.

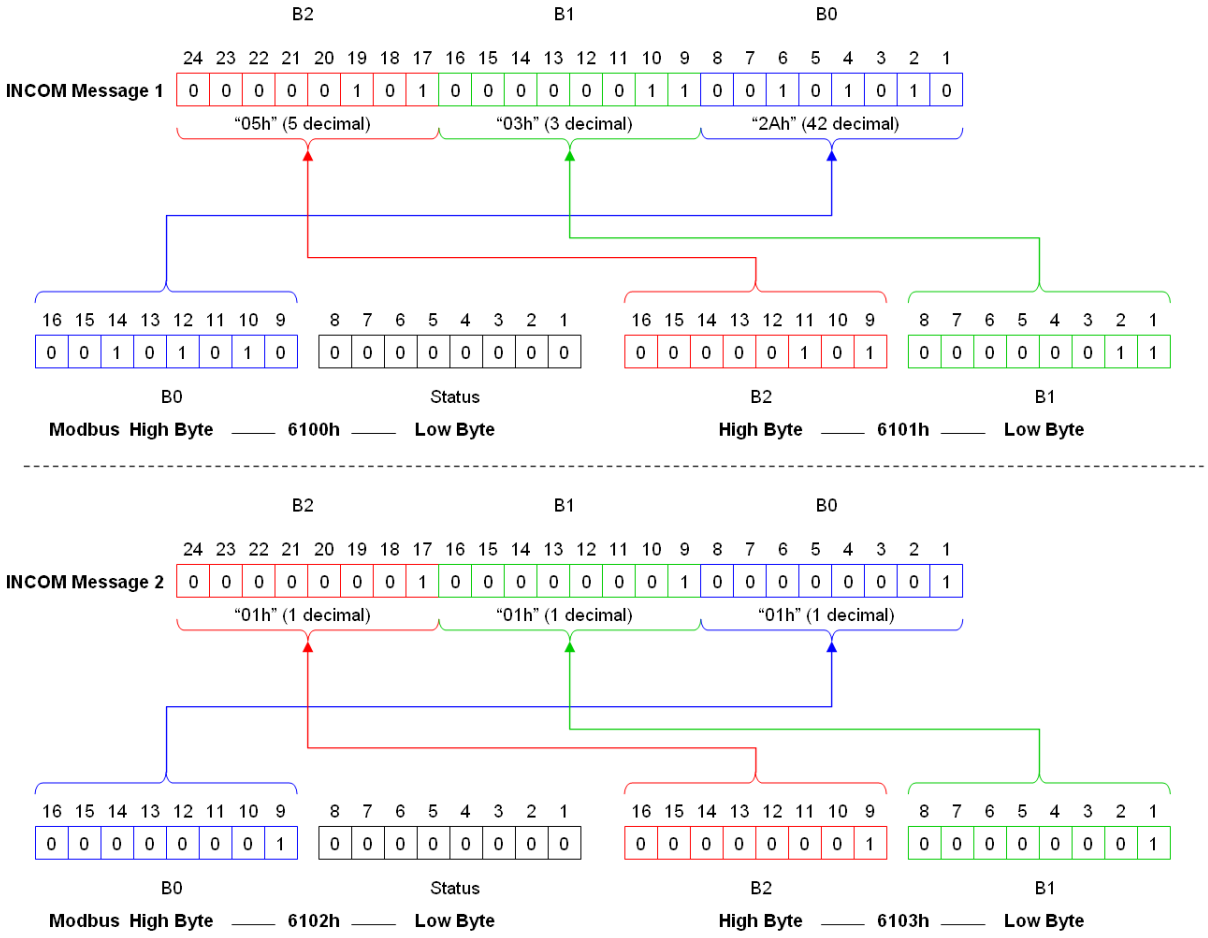

Figure 5

Notice that the first message returned consists of two 16-bit Modbus registers, 6100h and 6101 or 32-bits total. Since INCOM only transmits 24-bits of data per message, the remaining 8-bits are used as a status message sent by the MMINT or MPONI to advise whether the INCOM data retrieved from the ATC was valid or not. A zero in this status byte (low byte of the first of the two register pair) indicates no errors. This status byte is not used in our checksum calculation, although should it be non-zero, we would want to discard this data anyway as it would be invalid for different reasons (bad communications detected between MMINT/MPONI and ATC).

In this example, all status bytes are zero, so we begin our checksum calculation one byte at a time as follows:

1. Message 1, 6100 hex offset from beginning of Modbus Holding register table, high byte = 2A hex (42 decimal) – rolling checksum = 42 decimal
2. Message 1, 6101 hex offset from beginning of Modbus Holding register table, high byte = 05 hex (5 decimal) – rolling checksum = 47 decimal
3. Message 1, 6101 hex offset from beginning of Modbus Holding register table, low byte = 03 hex (3 decimal) – rolling checksum = 50 decimal

4. Message 2, 6102 hex offset from beginning of Modbus Holding register table, high byte = 1 hex (1 decimal) – rolling checksum = 51 decimal
5. Message 2, 6103 hex offset from beginning of Modbus Holding register table, high byte = 1 hex (1 decimal) – rolling checksum = 52 decimal
6. Message 2, 6103 hex offset from beginning of Modbus Holding register table, low byte = 1 hex (1 decimal) – rolling checksum = 53 decimal

At this point after processing 3 bytes in our setpoint table from the first two messages, the rolling checksum equals 53 decimal.

When all the [B2], [B1] and [B0] values for each of the 42 data messages are summed, this checksum is then compared with the 43rd message which contains the value of checksum computed by the ATC. That 16-bit value is stored in bytes [B1] [B0] of message 43. For an additional check, the ATC computes the 1's complement of the low byte [B0] and places this value in the 43rd message as [B2]

Table 2

Message	Hex			(Base 16)	(Base 10)	
43	6154	B0	H	FD	253	Checksum (sum of previous 42 messages) low byte
		S	L	00	0	status
	6155	B2	H	02	2	Complement of Checksum low byte
		B1	L	07	7	Checksum (sum of previous 42 messages) high byte

4. Procedure to Read Setpoint Buffer

The steps are as follows:

1. Write the value $C3AB_{16}$ into MMINT (or MPONI[1]) register 6000_{16} and the value 9001_{16} into register 6001_{16} using one Modbus write register (function code 16 decimal) instruction. C3AB corresponds to sending the 3 C 9 INCOM command with the expectation that 43 INCOM messages will be returned.

```
'***Example***
' send 3 C 9 command to INCOM address 001
'[INST] = 3 (0011), [COMM] = C (1100), [SCOM] = 9 (1001)
'3 C 9 is a control message and it expects 43 messages in reply
'43 decimal is 101011 binary
'when combined with the most significant bit being 1 (for control msg)
'the result is 0xAB = 1010 1011

'0x6000 15 14 13 12   11 10 09 08   07 06 05 04   03 02 01 00
'       --[COMM]--    --[INST]--    CD  R  [# of msgs expected]
'        1  1  0  0    0  0  1  1    1  0  1  0    1  0  1  1
'            C             3             A             B
'0x6001 15 14 13 12   11 10 09 08   07 06 05 04   03 02 01 00
'       --[SCOM]-- /  -[12-bit INCOM address of device]-
'        1  0  0  1    0  0  0  0    0  0  0  0    0  0  0  1
'            9             0             0             1
'So, 0x6000 will be loaded with 0xC3AB
'and 0x6001 will be loaded with 0x9001
```

[1] As mentioned earlier, the MPONI is not recommended when reading the setpoint buffer

2. If Modbus write command was successful, then wait for 50 ms to allow the MMINT time to send the message onto the INCOM slave.

```
Do While iLoopCount < giMaxLoopCount - 1
            If MB_Write(1, &H6000, mb_buffer, 2) Then
                'message transmitted okay
                frmMain.lbl_MB_Xsum.Text = "Valid"
                frmMain.lbl_MB_Xsum.ForeColor = Color.Black
'wait to allow that first message to be processed
                Pause(50)   '50 ms
```

3. Read the Modbus buffer beginning at 6000_{16}. The number of registers to read depends on the particular setpoint buffer. For example, the ATC-800 setpoint buffer contains 43 24-bit setpoints. This will be returned as 86 16-bit Modbus registers.

```
'now retrieve data from address &h6100
                If MB_Read(1, &H6100, mb_buffer, (2 * NumMsgReplyExpected)) Then
                    'message received okay
                    frmMain.lbl_MB_Xsum.Text = "Valid"
                    frmMain.lbl_MB_Xsum.ForeColor = Color.Black
                    bEFlag = False
                    Exit Do
                Else
                    'failed somehow
                    frmMain.lbl_MB_Xsum.Text = "Invalid"
                    frmMain.lbl_MB_Xsum.ForeColor = Color.Red
                    g_MB_errors += 1
                    bEFlag = True
                End If
```

5. Procedure to Write Setpoint Buffer

Once the setpoint buffer has been retrieved by the Modbus master and the appropriate changes are made, the steps to follow are:

1. Calculate the new Setpoint Buffer checksum
    a. Divide this value by 256 and ignore any remainder – save this in high byte of checksum
    b. Take high byte of checksum and multiply by 256 and subtract this product from complete checksum – save this in the low byte of the checksum.
    c. Take the 1's complement of the low byte of the checksum

```
'load checksum into appropriate registers
        SPBuffer(174) = newSPXsum \ 256                     'high byte of checksum
        SPBuffer(171) = newSPXsum - SPBuffer(174) * 256     'low byte
        SPBuffer(173) = Not SPBuffer(171)                   'complement of low byte
```

2. Write the value $F381_{16}$ into MMINT (or MPONI[1]) register 6000_{16} and the value 9001_{16} into register 6001_{16} using one Modbus write register (function code 16 decimal) instruction. This will tell the MMINT to expect 43 more messages.

```
'pass-through protcol requires that each message be transmitted separate
        'First, send 3 F 9
        mb_buffer = {&HF381, &H9001}    '3 F 9 and 9000 + INCOM add
        If MB_Write(1, &H6000, mb_buffer, 2) Then
            'INCOM 3 F 9 control message transmitted okay
```

3. Send the first of the 43 messages by writing the first 32 bits into Modbus registers 6000_{16} and 6001_{16}.

```
For i = 1 To 43
        'download 42 setpoints.  43rd message is checksum of previous 42 setpoints
        'each message begins on a 4-byte boundary since each is 32-bits (2 Modbus
        'registers) long
        byte_pointer = i * 4 - 1    'msg 1 ->3, 4, 5, 6; msg 2 -> 7, 8, 9, 10
        'mb_buffer(0) = [B0]01  mb_buffer(1) = [B2][B1]
        mb_buffer(0) = SPBuffer(byte_pointer) * 256 + 1
        mb_buffer(1) = SPBuffer(byte_pointer + 2) * 256 + SPBuffer(byte_pointer + 3)
        'transmit INCOM data message
        If MB_Write(1, &H6000, mb_buffer, 2) Then
                'successful write
               Else
                   'failed to write
                   frmMain.lbl_MB_Xsum.Text = "Invalid"
                   frmMain.lbl_MB_Xsum.ForeColor = Color.Red
                   frmMain.lbl_MB_total_errors.Text = Str(g_MB_errors)
                   frmMain.rtb_Advisory.AppendText(vbCrLf + "MB Write to 0x6000, Msg:" + i _
                     + " failed" + vbCrLf)
                   Exit Sub    'exit early since the message will be discarded at INCOM
                               'device anyway
               End If
Next
```

4. Repeat step 3 for each message.

6. Visual Basic Example Programs

While Modbus master routines are available in a variety of versions of Microsoft's Visual Basic, the newest version (as of the date of this document) is Visual Basic 2010. The "Express" version of this program is available free of charge[2] for both personal and business use.

The following code examples are written in VB2010 Express.

6.1. MB_Read

This routine reads a block of registers from a Modbus slave such as the MMINT or MPONI using Modbus function code 03.

Inputs:

mb-add	Modbus slave address (1-247 typically)
start	Starting register address in Modbus slave (0 is first address)
length	Number of 16-bit registers to read

[1] As mentioned earlier, the MPONI is not recommended when reading the setpoint buffer

[2] http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-basic-express

Outputs:

data_2_read	Buffer to place received data
MB_Read	Status returned (1: good, 0: failed)

```
Public Function MB_Read(ByVal mb_add As Byte, ByVal start As Integer, ByRef data_2_read As UInt16(),
ByVal length As Integer) As Byte
        'read from slave using Modbus function code 03
        'Send:
        '[Slave Add][Func 03][HB start add][LB start add][HB # regs to write][LB of # regs to write][LB
CRC][HB CRC]
        'Response:
        '[Slave Add][Func 03 dec AND error code][Byte Count][HB D1][Lb D1][HB D2][LB D2]...[HB Dn][LB
Dn][LB CRC][HB CRC]

        Dim ModbusMsg(256) As Byte
        Dim CRC As UInt16
        Dim MsgStr As String = ""
        Dim i, x As Integer
        Dim temp_string As String
        byte_counter = 0 'clear

        g_bCommActive = True    'comm active with this device (might be redundant, but include just to be
sure)

        ModbusMsg(0) = mb_add
        ModbusMsg(1) = 3
        ModbusMsg(2) = start \ 256
        ModbusMsg(3) = start - ModbusMsg(2) * 256
        ModbusMsg(4) = length \ 256
        ModbusMsg(5) = length - ModbusMsg(4) * 256

        'compute the CRC
        CRC = ModCRC(ModbusMsg, 6)
        'must invert byte order HB->LB and LB->HB
        ModbusMsg(7) = CRC \ 256
        ModbusMsg(6) = CRC - ModbusMsg(7) * 256

        'convert byte array to hex string
        For i = 0 To (8) - 1
            temp_string = (Convert.ToString(ModbusMsg(i), 16).PadLeft(2, "0"c).PadRight(3, " "c))
            'padded = StrDup(2 - Len(temp_string), "0") & temp_string
            'MsgStr = MsgStr + padded
            MsgStr = MsgStr + temp_string
        Next
        WriteByte(ModbusMsg, 8, True)

        Pause(20)    'pause 50 ms
        If array_count <> 0 Then
            Pause(2)   'wait 20 ms
            x = array_count     'read number of characters
            Pause(1)    'wait 10 ms
            Do Until array_count = x
                x = array_count 'read number of characters
                Pause(1)    'wait 10 ms
            Loop
        End If

        'now parse the reply stored in global array data_buffer
        If VerifyCRC(data_buffer, byte_counter) Then
            'CRC on reply is good
            MB_Read = 1
        Else
```

```
            'CRC on reply failed
            MB_Read = 0
            g_MB_errors += 1
        End If
        Pause(5)
        If array_count <> 0 Then
            Pause(2)   'wait 20 ms
            x = array_count     'read number of characters
            Pause(1)    'wait 10 ms
            Do Until array_count = x
                x = array_count 'read number of characters
                Pause(1)    'wait 10 ms
            Loop
        End If

    End Function
```

6.2. MB_Write

This routine writes a block of registers to a Modbus slave such as the MMINT or MPONI using the Modbus function code 10h (16 decimal). Note that with the Eaton implementation of the Modbus write 10h function when using the MMINT or MPONI, the maximum write length is limited to two 16-bit registers. Longer blocks of registers must be divided into multiple write messages and sent separately.

Input:

mb-add	Modbus slave address (1-247 typically)
start	Starting register address in Modbus slave (0 is first address)
data_2_read	Buffer to be written to Modbus slave
length	Number of 16-bit registers to write

Output:

MB_Write	Status of CRC in acknowledgement from Modbus slave (1: good, 0: bad)

```
Public Function MB_Write(ByVal mb_add As Byte, ByVal start As Integer, ByVal data_2_write As UInt16(),
ByVal length As Integer) As Byte

        'write to slave using Modbus function code 16 (10 hex)
        'mb_add     Modbus address of slave
        'start      starting register address to write
        'data_2_write   16 bit data, written as 2x8-bit bytes
        'length     number of 16 bit data registers to write

        'Send:
        '[Slave Add][Func 16 dec, 10 hex][HB start add][LB start add][HB # regs to write][LB of # regs to
write][Byte Count][HB D1][Lb D1][HB D2][LB D2]...[HB Dn][LB Dn][LB CRC][HB CRC]

        'Response:
        '[Slave Add][Func 16 dec AND error code][HB start add][LB start add][HB # regs to write][LB of #
regs to write][LB CRC][HB CRC]

        Dim ModbusMsg(256) As Byte
        Dim CRC As UInt16
        Dim MsgStr As String = ""
        Dim i, x As Integer
        Dim temp_string As String
        byte_counter = 0 'clear

        ModbusMsg(0) = mb_add
```

```
        ModbusMsg(1) = 16   '10 hex
        ModbusMsg(2) = start \ 256
        ModbusMsg(3) = start - ModbusMsg(2) * 256
        ModbusMsg(4) = length \ 256
        ModbusMsg(5) = length - ModbusMsg(4) * 256
        ModbusMsg(6) = length * 2
        For i = 0 To length - 1
            ModbusMsg(7 + 2 * i) = data_2_write(i) \ 256
            ModbusMsg(8 + 2 * i) = data_2_write(i) - ModbusMsg(7 + 2 * i) * 256
        Next

        'compute the CRC
        CRC = ModCRC(ModbusMsg, 7 + 2 * length)
        'must invert byte order HB->LB and LB->HB
        ModbusMsg(8 + 2 * length) = CRC \ 256
        ModbusMsg(7 + 2 * length) = CRC - ModbusMsg(8 + 2 * length) * 256

        'convert byte array to hex string
        For i = 0 To (9 + 2 * length) - 1
            temp_string = (Convert.ToString(ModbusMsg(i), 16).PadLeft(2, "0"c).PadRight(3, " "c))
            'padded = StrDup(2 - Len(temp_string), "0") & temp_string
            'MsgStr = MsgStr + padded
            MsgStr = MsgStr + temp_string
        Next
        WriteByte(ModbusMsg, 9 + 2 * length, True)

        Pause(6)
        If array_count <> 0 Then
            Pause(2)   'wait 20 ms
            x = array_count     'read number of characters
            Pause(1)    'wait 10 ms
            Do Until array_count = x
                x = array_count 'read number of characters
                Pause(1)    'wait 10 ms
            Loop
        End If

        'now parse the reply stored in global array data_buffer
        If VerifyCRC(data_buffer, byte_counter) Then
            'CRC on reply is good
            MB_Write = 1
        Else
            'CRC on reply failed
            MB_Write = 0
            g_MB_errors += 1
        End If

    End Function
```

6.3. ModCRC

This function calculates the CRC-16 checksum over the length of characters transmitted in the Modbus message. This checksum is added as the last two bytes in the message.

Input:

Buffer()	Array holding the Modbus message
length	Number of bytes in array buffer

Output:

ModCRC	16 bit CRC-16

```
    'Copyright Richard L. Grier, 2006
```

```
    '-----------------------------------------------
    Public Function ModCRC(ByVal Buffer() As Byte, ByVal length As Integer) As Integer
        '-----------------------------------------------

        ' returns the MODBUS CRC of buffer
        Dim CRC1 As Long
        Dim I As Integer
        Dim J As Integer
        Dim K As Long

        CRC1 = &HFFFF ' init CRC
        'For I = 0 To UBound(Buffer) - 1 ' each byte
        For I = 0 To length - 1 ' each byte
            CRC1 = CRC1 Xor Buffer(I)
            For J = 0 To 7 ' for each bit in byte
                K = CRC1 And 1 ' bit 0 value
                CRC1 = ((CRC1 And &HFFFE) / 2) And &H7FFF ' Shift right with 0 ms bit
                If K > 0 Then CRC1 = CRC1 Xor &HA001
            Next J
        Next I
        ModCRC = CRC1
    End Function
```

6.4. VerifyCRC

This function compares the CRC appended on end of the Modbus message with a separate calculation of the CRC. The two CRC values are compared and if same, then a valid flag is set. If the two CRC values do not match, the assumption is made that the message was corrupted during transmission and a flag is set indicating the message is invalid and should be discarded.

Input:

buffer	Buffer containing the Modbus message
mb_count	Number of bytes in the Modbus message buffer

Output:

VerifyCRC	Status (0: failed test, 1: passed test)

```
Public Function VerifyCRC(ByVal buffer As Byte(), ByVal mb_count As Integer)
        'recalculate CRC over message and verify it matches CRC on message
        Dim CRC As UInt16
        Dim CRC1 As Byte
        Dim CRC2 As Byte

        If mb_count = 0 Then    'nothing received
            VerifyCRC = 0   'failed test
            frmMain.rtb_Advisory.AppendText("No response received to MB request" + vbCrLf)
            Exit Function
        End If

        CRC = ModCRC(buffer, mb_count - 2)
        CRC1 = CRC \ 256
        CRC2 = CRC - CRC1 * 256

        If buffer(mb_count - 1) <> CRC1 Then
            VerifyCRC = 0   'failed test
            Exit Function
        ElseIf buffer(mb_count - 2) <> CRC2 Then
            VerifyCRC = 0   'failed test
            Exit Function
```

```
        Else
            VerifyCRC = 1   'both bytes match.  passed test
        End If

    End Function
```

6.5. SendINCOMCtrlMsg

Transmit a single INCOM control message as an embedded message within a Modbus. Referring to the INCOM/IMPACC protocol manual[1] the message consists of three parts:

[INST] [COMM] [SCOM]

A typical control message is the “Transmit Setpoints Buffer – 3 C 9”

INST = 3 (0011 binary)

COMM = C (1100 binary)

SCOM = 9 (1001 binary)

The MMINT/MPONI Modbus write protocol requires that these values be loaded into Modbus registers 6000h and 6001h as follows:

6000h bits 15-12 contain [COMM]

6000h bits 11-8 contain [INST]

6000h bit 7 contains 1 (meaning this is a control message)

6000h bit 6 contains 0 (reserved for future use)

6000h bits 5-0 contains binary number equal to the number of messages expected to returned

6001h bits 15-12 contains [SCOM]

6001h bits 11-0 contains the 12-bit binary address of the INCOM slave (1-4095)

The MMINT/MPONI receives this message, extracts the INCOM payload ([INST] [COMM] [SCOM]), transmits this payload to the INCOM device and waits for the response. The response is then stored in separate registers (address 6100h). This function retrieves the returned data from those registers and places the raw data in the Visual Basic global variable mb_buffer.

Input:

INCOMAdd	INCOM slave address (1-4095 decimal, 000 0001 to 111 1111 binary)
INST	INCOM instruction field
COMM	INCOM command field
SCOM	INCOM sub-command field
NumMsgReplyExpected	Number of INCOM 24-bit messages expected in response
INCOMResponse	Array containing INCOM messages

Output:

bEFlag	Error flag (False: no error, True: error)

[1] http://www.eaton.com/ecm/groups/public/%40pub/%40electrical/documents/content/1030709222496.pdf

SendINCOMCtrlMsg Status message (0: error, 1: no error)

```
Public Function SendINCOMCtrlMsg(ByVal INCOMAdd As UInt16, ByVal INST As Byte, ByVal COMM As Byte, ByVal
SCOM As Byte, ByVal NumMsgReplyExpected As UInt16, ByRef INCOMResponse As Byte())

        'the MMINT/MPONI pass-through protocol requires that we write INCOM msg to a two register pair
        '0x6000 and 0x6000 with the following format for a control message

        'Control Message:
        '0x6000 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
        '       --[COMM]-- /--[INST]--  CD R [# of msgs expected]
        'CD = 1: xmit INCOM control message
        'R = reserved (always 0)

        '0x6001 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
        '       --[SCOM]-- /-[12-bit INCOM address of device]-

        '***Example***
        'send 3 C 9 command to INCOM address 001
        '[INST] = 3 (0011), [COMM] = C (1100), [SCOM] = 9 (1001)
        '3 C 9 is a control message and it expects 43 messages in reply
        '43 decimal is 101011 binary
        'when combined with the most significant bit being 1 (for control msg)
        'the result is 0xAB = 1010 1011

        '0x6000 15 14 13 12   11 10 09 08   07 06 05 04   03 02 01 00
        '       --[COMM]--    --[INST]--    CD  R  [# of msgs expected]
        '        1  1  0  0    0  0  1  1    1  0  1  0    1  0  1  1
        '            C           3             A            B
        '0x6001 15 14 13 12   11 10 09 08   07 06 05 04   03 02 01 00
        '       --[SCOM]-- /  -[12-bit INCOM address of device]-
        '        1  0  0  1    0  0  0  0    0  0  0  0    0  0  0  1
        '            9             0            0            1
        'So, 0x6000 will be loaded with 0xC3AB
        'and 0x6001 will be loaded with 0x9001


        Dim mb_buffer As UInt16() = {0, 0}
        Dim iLoopCount As Integer
        Dim bEFlag As Boolean = False

        'preload with INCOM values
        mb_buffer(0) = 128 + (NumMsgReplyExpected) + (INST * 256) + (COMM * 4096)
        mb_buffer(1) = INCOMAdd + (SCOM * 4096)

        Do While iLoopCount < giMaxLoopCount - 1
            If MB_Write(1, &H6000, mb_buffer, 2) Then
                'message transmitted okay
                frmMain.lbl_MB_Xsum.Text = "Valid"
                frmMain.lbl_MB_Xsum.ForeColor = Color.Black

                'wait to allow that first message to be processed
                Pause(50)   '200 ms

                'now retrieve data from address &h6100
                If MB_Read(1, &H6100, mb_buffer, (2 * NumMsgReplyExpected)) Then
                    'message received okay
                    frmMain.lbl_MB_Xsum.Text = "Valid"
                    frmMain.lbl_MB_Xsum.ForeColor = Color.Black
                    bEFlag = False
                    Exit Do
                Else
                    'failed somehow
                    frmMain.lbl_MB_Xsum.Text = "Invalid"
```

```
                frmMain.lbl_MB_Xsum.ForeColor = Color.Red
                g_MB_errors += 1
                bEFlag = True
            End If

        Else
            'failed somehow
            frmMain.lbl_MB_Xsum.Text = "Invalid"
            frmMain.lbl_MB_Xsum.ForeColor = Color.Red
            g_MB_errors += 1
            frmMain.lbl_MB_total_errors.Text = Str(g_MB_errors)
            If frmMain.lbl_MB_total_errors.Text > 0 Then
                frmMain.lbl_MB_total_errors.ForeColor = Color.Red
            Else
                frmMain.lbl_MB_total_errors.ForeColor = Color.Black
            End If
            SendINCOMCtrlMsg = 1

        End If
        iLoopCount += 1
        UpdateCommStatus("Attempt:" + Str(iLoopCount + 1))
    Loop
    MoveINCOMResponse(INCOMResponse, NumMsgReplyExpected * 4)
    If bEFlag Then
        SendINCOMCtrlMsg = 0
    Else
        SendINCOMCtrlMsg = 1
    End If
End Function
```

6.6. SendINCOMDataMsg

INCOM data messages are used either as replies to a control message or as subsequent messages sent by an INCOM master after that master has sent a control message. For example, if the INCOM master is sending a setpoint buffer to an INCOM slave, it will begin by sending the 3 C 9 control command and then follow it up with 43 data messages of the form that follows:

Input:

B2 INCOM Data Message Byte 2 of 24-bit INCOM data message

B1 INCOM Data Message Byte 1 of 24-bit INCOM data message

B0 INCOM Data Message Byte 0 of 24-bit INCOM data message

Output:

```
Public Sub SendINCOMDataMsg(ByRef B2 As Byte, ByRef B1 As Byte, ByRef B0 As Byte)

    'the MMINT/MPONI pass-through protocol requires that we write INCOM msg to a two register pair
    '0x6000 and 0x6000 with the following format for a data message
    'Data Message:
    '0x6000 bit 15-8    INCOM [B0]
    '       bit 7       C/D bit 1: INCOM control msg, 0: INCOM data msg
    '       bit 6       0 (reserved)
    '       bits 5-0    # of INCOM messages expected in reply to this message
    '0x6001 bits 15-8   INCOM [B2]
    '       bits 7-0    INCOM [B1]

    'written out bit-wise:
    '0x6000 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
    '        -------[B0]----------   0  R [# of msgs expected]
    'bit 07 = 0: since this is a xmit INCOM data messsage
```

```
        'R = reserved (always 0)

        '0x6001 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
        '       -------[B2]----------    -------[B0]----------

        Dim mb_buffer As UInt16()
        Dim iLoopCount As Integer
        'first check if control or data message
        mb_buffer = {&HC3AB, &H9001}

        Do While iLoopCount < giMaxLoopCount - 1
            If MB_Write(1, &H6000, mb_buffer, 2) Then
                'message transmitted okay
                frmMain.lbl_MB_Xsum.Text = "Valid"
                frmMain.lbl_MB_Xsum.ForeColor = Color.Black

                'wait to allow that first message to be processed
                Pause(50)   '200 ms

                'now retrieve data from address &h6100
                If MB_Read(1, &H6100, mb_buffer, 86) Then
                    'message received okay
                    frmMain.lbl_MB_Xsum.Text = "Valid"
                    frmMain.lbl_MB_Xsum.ForeColor = Color.Black
                    Exit Do
                Else
                    'failed somehow
                    frmMain.lbl_MB_Xsum.Text = "Invalid"
                    frmMain.lbl_MB_Xsum.ForeColor = Color.Red
                    g_MB_errors += 1
                End If

            Else
                'failed somehow
                frmMain.lbl_MB_Xsum.Text = "Invalid"
                frmMain.lbl_MB_Xsum.ForeColor = Color.Red
                g_MB_errors += 1
                frmMain.lbl_MB_total_errors.Text = Str(g_MB_errors)
                If frmMain.lbl_MB_total_errors.Text > 0 Then
                    frmMain.lbl_MB_total_errors.ForeColor = Color.Red
                Else
                    frmMain.lbl_MB_total_errors.ForeColor = Color.Black
                End If

            End If
            iLoopCount += 1
            UpdateCommStatus("Attempt:" + Str(iLoopCount + 1))
        Loop
    End Sub
```

6.7. ReadFastStatus

Send a 3 0 0 INCOM Fast Status request to an INCOM slave device. The response of the fast status message is device dependent, but the comments in the code below show the interpretation of the fast status response for the ATC-600 and -800 transfer switch controllers.

Input:

INCOMAdd	INCOM 12-bit address of slave

Output:

values()	Array variable that contains the returned data from the fast status request The meaning of these values depends on the INCOM slave device

```
Public Function ReadFastStatus(ByRef INCOMAdd As UInt16, ByRef values As UInt16()) As Boolean

        Dim msg_temp(100) As Byte
        Dim mb_buffer As UInt16() = {0, 0}
        Dim iLoopCount As Integer

        Do While iLoopCount < giMaxLoopCount - 1
            If Not (SendINCOMCtrlMsg(INCOMAdd, 3, 0, 0, 1, msg_temp)) Then
                'message transmitted okay
                frmMain.lbl_MB_Xsum.Text = "Valid"
                frmMain.lbl_MB_Xsum.ForeColor = Color.Black
                'msg_temp(0) through msg_temp(3) contains the Fast Status data returned by the device
                'msg_temp(0) = [B0] of Msg 0
                'msg_temp(1) = status byte of INCOM Msg 0
                'msg_temp(2) = [B2] of Msg 0
                'msg_temp(3) = [B1] of Msg 0
                '           [B2]         |          [B1]           |           [B0]
                'S7 S6 S5 S4 S3 S2 S1 S0 | P5 P4 P3 P2 P1 P0 V3 V2 | V1 V0 D5 D4 D3 D2 D1 D0
                'sample reply from ATC
                '0  0  0  0  1  1  1  1    0  1  0  1  0  1  0  0    1  1  0  0  0  1  0  0
                '    [0]       [F]           [5]         [4]           [C]         [4]

                'S7 S6
                '0  0   On good source (see S2, S5)
                '0  1   Generator Start
                '1  0   Tranferred (see S2, S5)
                '1  1   Alarm
                'S5 1-Source 2 is connected
                'S4 1-Powered on since last fast status read attempt
                'S3 1-Unread time-stamped buffer available
                'S2 1-Source 1 is connected
                'S1 1-Source 2 is available
                'S0 1-Source 1 is available
                'P5-P0 product ID (ATC - 21)
                'V3-V0 communication software version (ATC - 0 [initial phase 2 version], 10 TDEN
extended to 8 hours)
                'D5-D0 division code (ATC - 4)

                values(0) = (msg_temp(2) And &HC0) \ 64     'S7S6
                values(1) = (msg_temp(2) And &H3F)          'S5-S0
                values(2) = (msg_temp(3) And &HFC) \ 4      'P5-P0
                values(3) = (msg_temp(3) And 3) + (msg_temp(0) And &HC0) \ 64   'V3-V0
                values(4) = (msg_temp(0) And &H3F)          'D3-D0

                Exit Do
            Else
                'failed somehow
                frmMain.lbl_MB_Xsum.Text = "Invalid"
                frmMain.lbl_MB_Xsum.ForeColor = Color.Red
                g_MB_errors += 1
                frmMain.lbl_MB_total_errors.Text = Str(g_MB_errors)
                If frmMain.lbl_MB_total_errors.Text > 0 Then
                    frmMain.lbl_MB_total_errors.ForeColor = Color.Red
                Else
                    frmMain.lbl_MB_total_errors.ForeColor = Color.Black
                End If
                ReadFastStatus = 0

            End If
            iLoopCount += 1
            UpdateCommStatus("Attempt:" + Str(iLoopCount + 1))
        Loop

    End Function
```

6.8. ReadSPBuffer

Read the setpoint buffer from an INCOM slave. This version of the routine is designed to read the setpoint buffer from an ATC-600 or -800 transfer switch controller. The routine could be modified to be more general purpose.

Input:

giMaxLoopCount global variable giving number of messages expected in SP buffer

Output:

mb_buffer Modbus data returned

```
Public Sub ReadSPBuffer()

        Dim mb_buffer As UInt16()
        Dim Xsum As UInt16
        Dim iLoopCount As Integer

        'g_bCommActive = True    'tell other threads that we are using the comm port to talk to remote
device

        UpdateCommStatus("Reading setpoints from ATC")

        'clear out message on screen that says the SP buffer checksum is valid or invalid
        frmMain.lbl_Xsum.Text = "(unknown)"
        frmMain.lbl_Xsum.ForeColor = Color.Red
        frmMain.lbl_MB_Xsum.Text = "(unknown)"
        frmMain.lbl_MB_Xsum.ForeColor = Color.Red

        mb_buffer = {&HC3AB, &H9001}

        Do While iLoopCount < giMaxLoopCount - 1
            If MB_Write(1, &H6000, mb_buffer, 2) Then
                'message transmitted okay
                frmMain.lbl_MB_Xsum.Text = "Valid"
                frmMain.lbl_MB_Xsum.ForeColor = Color.Black

                'wait to allow that first message to be processed
                Pause(50)   '200 ms

                'now retrieve data from address &h6100
                If MB_Read(1, &H6100, mb_buffer, 86) Then
                    'message received okay
                    frmMain.lbl_MB_Xsum.Text = "Valid"
                    frmMain.lbl_MB_Xsum.ForeColor = Color.Black
                    Exit Do
                Else
                    'failed somehow
                    frmMain.lbl_MB_Xsum.Text = "Invalid"
                    frmMain.lbl_MB_Xsum.ForeColor = Color.Red
                    g_MB_errors += 1
                End If

            Else
                'failed somehow
                frmMain.lbl_MB_Xsum.Text = "Invalid"
                frmMain.lbl_MB_Xsum.ForeColor = Color.Red
                g_MB_errors += 1
                frmMain.lbl_MB_total_errors.Text = Str(g_MB_errors)
                If frmMain.lbl_MB_total_errors.Text > 0 Then
                    frmMain.lbl_MB_total_errors.ForeColor = Color.Red
                Else
```

```
                frmMain.lbl_MB_total_errors.ForeColor = Color.Black
            End If

        End If
        iLoopCount += 1
        UpdateCommStatus("Attempt:" + Str(iLoopCount + 1))
    Loop

    'now verify that the setpoint buffer checksum matches what was sent
    If VerifyINCOMXsum(data_buffer, byte_counter) Then
        'valid INCOM xsum over 3 C 9 buffer
        frmMain.lbl_Xsum.Text = "Valid"
        frmMain.grp_PreferredSource.Enabled = True
        frmMain.lbl_Xsum.ForeColor = Color.Black
    Else
        'invalid INCOM xsum
        frmMain.lbl_Xsum.Text = "Invalid"
        frmMain.lbl_Xsum.ForeColor = Color.Red
        g_MB_errors += 1
        UpdateCommStatus("Unable to retrieve setpoints")
        Exit Sub
    End If

    'at this point we have downloaded a valid SP buffer.  Save it
    SaveSPBuffer()

    'test *** calculate the SP Buffer Xsum
    Xsum = CalcSPBufferXsum(SPBuffer, byte_counter)

    'update error counter on screen
    frmMain.lbl_MB_total_errors.Text = Str(g_MB_errors)

    'display INCOM Xsum on screen
    frmMain.lbl_IXS.Text = Xsum

    'now print the data from the buffer to the screen
    print_data(data_buffer, array_count)

    'save the length of the SP buffer for later
    g_iSPArray_length = byte_counter

    'g_bCommActive = False   'tell other threads that the comm port is free to use

    UpdateCommStatus("")

End Sub
```

6.9. SPWrite

A routine to write the setpoint buffer back to the INCOM slave using the INCOM pass-through functionality of the MMINT/MPONI. As with the previous function (ReadSPBuffer), this routine is hard coded to support writing to the ATC-600 or -800 setpoint buffer. The routine could be modified to be more general purpose and support any device that uses the 3 F 9 INCOM (Receive Multi-Block Setpoint Data Packet).

Input:

SPBuffer VB buffer containing the setpoints to be transmitted

Output:

```
Public Sub SPWrite()
    'write back setpoints to ATC using 3 F 9 INCOM pass-through command
    Dim mb_buffer() As UInt16
```

```
        Dim newSPXsum As UInt16
        Dim i As Integer
        Dim byte_pointer As Integer

        'update the checksum after changes have been made
        newSPXsum = CalcSPBufferXsum(SPBuffer, g_iSPArray_length)
        UpdateCommStatus("Writing ATC setpoints")

        'load checksum into appropriate registers
        SPBuffer(174) = newSPXsum \ 256                     'high byte of checksum
        SPBuffer(171) = newSPXsum - SPBuffer(174) * 256   'low byte
        SPBuffer(173) = Not SPBuffer(171)                 'complement of low byte

        'pass-through protcol requires that each message be transmitted separate
        'First, send 3 F 9
        mb_buffer = {&HF381, &H9001}    '3 F 9 and 9000 + INCOM add
        If MB_Write(1, &H6000, mb_buffer, 2) Then
            'INCOM 3 F 9 control message transmitted okay
            frmMain.lbl_MB_Xsum.Text = "Valid"
            frmMain.lbl_MB_Xsum.ForeColor = Color.Black
            'now we need to send 43 additional INCOM data messages with updated setpoints
            For i = 1 To 43
                'download 42 setpoints.  43rd message is checksum of previous 42 setpoints
                'each message begins on a 4-byte boundary since each is 32-bits (2 Modbus registers) long
                byte_pointer = i * 4 - 1    'msg 1 ->3, 4, 5, 6; msg 2 -> 7, 8, 9, 10
                'mb_buffer(0) = [B0]01  mb_buffer(1) = [B2][B1]
                mb_buffer(0) = SPBuffer(byte_pointer) * 256 + 1
                mb_buffer(1) = SPBuffer(byte_pointer + 2) * 256 + SPBuffer(byte_pointer + 3)
                'transmit INCOM data message
                If MB_Write(1, &H6000, mb_buffer, 2) Then
                    'successful write
                Else
                    'failed to write
                    frmMain.lbl_MB_Xsum.Text = "Invalid"
                    frmMain.lbl_MB_Xsum.ForeColor = Color.Red
                    frmMain.lbl_MB_total_errors.Text = Str(g_MB_errors)
                    frmMain.rtb_Advisory.AppendText(vbCrLf + "MB Write to 0x6000, Msg:" + i + " failed" +
vbCrLf)
                    Exit Sub    'exit early since the message will be discarded at INCOM device anyway
                End If
            Next
        Else
            'initial INCOM control message failed
            frmMain.lbl_MB_Xsum.Text = "Invalid"
            frmMain.lbl_MB_Xsum.ForeColor = Color.Red
            frmMain.lbl_MB_total_errors.Text = Str(g_MB_errors)
            frmMain.rtb_Advisory.AppendText(vbCrLf + "Unable to write 3 F 9 control message to 0x6000" +
vbCrLf)
        End If

        're-read the SP Buffer to verify all values were downloaded
        'ReadSPBuffer()

        UpdateCommStatus("Waiting to confirm")

    End Sub
```

6.10. CalcSPBufferXsum

Computes the checksum setpoint values. This value then can be appended to the end of the setpoint buffer prior to transmitting to the INCOM slave. The INCOM slave will compare this calculated checksum with its own checksum calculated over the received setpoints. If the two checkpoints are the same, the INCOM slave will assume the setpoints are valid and will update its own setpoint values with these new values.

Input:

buffer Array buffer of setpoints

buffer_count number of values in setpoint buffer

Output:

```
Public Function CalcSPBufferXsum(ByVal buffer As Byte(), ByVal buffer_count As UInt16)
        'used to verify if the checksum over setpoints in 3 C 9 message matches what was sent
        Dim i As Integer
        Dim Xsum As UInt16

        'preload first byte into xsum
        Xsum = buffer(3)

        For i = 4 To buffer_count - 7   'subtract last two bytes for CRC, 4 for 43 msg
            'start with the 4th byte of the message (1st with real data)
            If (i Mod 4) Then
                Xsum = Xsum + buffer(i)
            End If
        Next

        CalcSPBufferXsum = Xsum

    End Function
```

6.11. VerifyINCOMXSum

Calculates the checksum of a setpoint buffer retrieved from an INCOM slave.

Input:

buffer VB array buffer containing retrieved setpoints

buffer_count Number of setpoints stored in the buffer array

Output:

VerifyINCOMXsum (0: invalid checksum, 1: valid checksum)

```
Public Function VerifyINCOMXsum(ByVal buffer As Byte(), ByVal buffer_count As Integer)
        'used to verify if the checksum over setpoints in 3 C 9 message matches what was sent
        Dim i As Integer
        Dim Xsum As UInt16
        Dim INCOMXsum As UInt16

        If buffer_count = 0 Then    'nothing received
            VerifyINCOMXsum = 0   'failed test
            Exit Function
        End If

        'preload first byte into xsum
        Xsum = buffer(3)

        For i = 4 To buffer_count - 7   'subtract last two bytes for CRC, 4 for 43 msg
            'start with the 4th byte of the message (1st with real data)
            If (i Mod 4) Then
                Xsum = Xsum + buffer(i)
            End If
        Next

        'now verify that the computed Xsum matches the Xsum in bytes 171 (low byte) and 174 (high byte)
```

```
        INCOMXsum = buffer(174) * 256 + buffer(171)
        If (INCOMXsum <> Xsum) Then
            'invalid checksum
            VerifyINCOMXsum = 0
        Else
            'valid checksum
            VerifyINCOMXsum = 1
        End If
    End Function
```

7. Appendix

The ATC-600 and -800 setpoint buffer cross-reference to Modbus register locations[1].

Byte Offset	MB Reg Offset	Message	Message Offset			Raw (Base 16)	
0						01	Modbus node address
1						03	Modbus function code
2						7A	Byte count returned
3	0	1	0	B0	H	2A	# addl msgs (42d, 2Ah)
4				S	L	00	status
5	1			B2	H	05	FW version
6				B1	L	03	FW revision
							Factory Set Options
7	2	2	1	B0	H	01	0. TDES enabled
8				S	L	00	status
9	3			B2	H	01	2. TDEN enabled
10				B1	L	01	1. TDNE enabled
11	4	3	2	B0	H	01	3. TDEC enabled
12				S	L	00	status
13	5			B2	H	01	5. S2 OF monitoring enabled
14				B1	L	01	4. S2 UF monitoring enabled
15	6	4	3	B0	H	01	6. S2 OV monitoring enabled
16				S	L	00	status
17	7			B2	H	03	8. Transfer Time Bypass PB (0: disabled, 1: TDEN bypass, 2: TDNE bypass, 3: TDEN/TDNE bypass either timer)
18				B1	L	01	7. S2 UV monitoring enabled
19	8	5	4	B0	H	01	9. User selectable preferred source enabled
20				S	L	00	status
21	9			B2	H	01	11. S1 UF monitoring enabled
22				B1	L	01	10. Plant Exerciser enabled
23	10	6	5	B0	H	01	12. S1 OF monitoring enabled
24				S	L	00	status
25	11			B2	H	01	14. Type of operation (0: automatic, 1: user-selectable)
26				B1	L	01	13. S1 OV monitoring enabled
27	12	7	6	B0	H	01	TDN enabled
28				S	L	00	status
29	13			B2	H	00	17. Pre-transfer signal on sub-net enabled
30				B1	L	00	16. TDN load sense (0: disabled, 1: enabled 2-30% of nominal V) Note: option 15 & 16 mutually exclusive
31	14	8	7	B0	H	00	18. Remote sequencing on sub-net enabled
32				S	L	00	status
33	15			B2	H	01	20. Overcurrent protection enabled
34				B1	L	01	19. Service Entrance (0: disabled [ignore go to neutral input], 1: enabled [respond to go to neutral input])
35	16	9	8	B0	H	00	21. Type of switch (0: stored energy, 1: motor driven)
36				S	L	00	status
37	17			B2	H	01	23. Load shed from S2 enabled
38				B1	L	00	22. User selectable PT ratio enabled
39	18	10	9	B0	H	01	24. S1 area protection enabled

[1] In the MMINT and MPONI, the data returned to the MMINT/MPONI as are result of a read request to an INCOM device is stored in Modbus registers beginning at 6100_{16}. Therefore, message 1 will be stored in 6100_{16} and 6101_{16}.

40		S	L	00	status
41	19	B2	H	00	26. Option 26 reserved
42		B1	L	00	25. 0: open transition only, 1: open or in-phase, 2: open or in-phase or closed transition

Byte Offset	MB Reg Offset	Message	Message Offset			Raw (Base 16)	
							User Programmable Options
43	20	11	10	B0	H	05	0. TDES timer - low byte (0-120 seconds)
44				S	L	00	status
45	21			B2	H	05	1. TDNE timer low byte (0-1800 seconds)
46				B1	L	00	0. TDES timer high byte
47	22	12	11	B0	H	00	1. TDNE timer high byte
48				S	L	00	status
49	23			B2	H	00	2. TDEN timer high byte (0-28800 s, 0 - 8 hours)
50				B1	L	05	2. TDEN timer low byte
51	24	13	12	B0	H	05	3. TDEC timer low byte (0-1800 seconds)
52				S	L	00	status
53	25			B2	H	58	4. Nominal frequency low byte (50 or 60, Hz x 10)
54				B1	L	00	3. TDEC timer high byte
55	26	14	13	B0	H	02	4. Nominal frequency high byte
56				S	L	00	status
57	27			B2	H	00	5. Nominal voltage high byte (110-600/50 Hz, 120-600/60Hz)
58				B1	L	78	5. Nominal voltage low byte
59	28	15	14	B0	H	6A	6. S1 UV drop out voltage low byte
60				S	L	00	status
61	29			B2	H	6A	7. S2 UV drop out level low byte
62				B1	L	00	6. S1 UV drop out voltage high byte
63	30	16	15	B0	H	00	7. S2 UV drop out level high byte
64				S	L	00	status
65	31			B2	H	00	8. S1 UV pick up level high byte
66				B1	L	6C	8. S1 UV pick up level low byte
67	32	17	16	B0	H	6C	9. S2 UV pick up level low byte
68				S	L	00	status
69	33			B2	H	7E	10. S1 OV drop out level low byte
70				B1	L	00	9. S2 UV pick up level high byte
71	34	18	17	B0	H	00	10. S1 OV drop out level high byte
72				S	L	00	status
73	35			B2	H	00	11. S2 OV drop out level high byte
74				B1	L	81	11. S2 OV drop out level low byte
75	36	19	18	B0	H	7C	12. S1 OV pick up level low byte
76				S	L	00	status
77	37			B2	H	7C	13. S2 OV pick up level low byte
78				B1	L	00	12. S1 OV pick up level high byte
79	38	20	19	B0	H	00	13. S2 OV pick up level high byte
80				S	L	00	status
81	39			B2	H	02	14. S1 UF drop out level high byte
82				B1	L	46	14. S1 UF drop out level low byte
83	40	21	20	B0	H	46	15. S2 UF drop out level low byte
84				S	L	00	status
85	41			B2	H	52	16. S1 UF pick up level low byte
86				B1	L	02	15. S2 UF drop out level high byte

Byte Offset	MB Reg Offset	Message	Message Offset			Raw (Base 16)	
87	42	22	21	B0	H	02	16. S1 UF pick up level high byte
88				S	L	00	status
89	43			B2	H	02	17. S2 UF pick up level high byte
90				B1	L	50	17. S2 UF pick up level low byte
91	44	23	22	B0	H	6A	18. S1 OF drop out level low byte
92				S	L	00	status
93	45			B2	H	6A	19. S2 OF drop out level low byte
94				B1	L	02	18. S1 OF drop out level high byte
95	46	24	23	B0	H	02	19. S2 OF drop out level high byte
96				S	L	00	status
97	47			B2	H	02	20. S1 OF pick up level high byte
98				B1	L	5E	20.S1 OF pick up level low byte
99	48	25	24	B0	H	5E	21. S2 OF pick up level low byte
100				S	L	00	status
101	49			B2	H	01	22. TDN with load sensing low byte (0: no, 1: yes)
102				B1	L	02	21. S2 OF pick up level high byte
103	50	26	25	B0	H	00	22. TDN with load sensing high byte
104				S	L	00	status
105	51			B2	H	00	23. TDN timer high byte
106				B1	L	03	23. TDN timer low byte
107	52	27	26	B0	H	1E	24. Load voltage decay threshold low byte (2-30%)
108				S	L	00	status
109	53			B2	H	01	25. Preferred source selection - low byte (0: none, 1:S1, 2:S2)
110				B1	L	00	24. Load voltage decay threshold high byte
111	54	28	27	B0	H	00	25. Preferred source selection - high byte
112				S	L	00	status
113	55			B2	H	00	26. Plant exerciser high byte
114				B1	L	01	26. Plant exerciser low byte (0: disabled, 1: enabled)
115	56	29	28	B0	H	00	27. Plant exerciser load transfer low byte (0: disabled, 1: enabled)
116				S	L	00	status
117	57			B2	H	02	28. Plant exerciser day of week low byte (1: Sun, 2: Monday, etc.)
118				B1	L	00	27. Plant exerciser load transfer high byte
119	58	30	29	B0	H	00	28. Plant exerciser day of week high byte
120				S	L	00	status
121	59			B2	H	00	29. Plant exerciser hour of day high byte
122				B1	L	0A	29. Plant exerciser hour of day low byte
123	60	31	30	B0	H	00	30. Plant exerciser minute low byte
124				S	L	00	status
125	61			B2	H	00	31. Manual retransfer mode low byte (0: auto, 1: pb return)
126				B1	L	00	30. Plant exerciser minute high byte
127	62	32	31	B0	H	00	31. Manual retransfer mode high byte
128				S	L	00	status
129	63			B2	H	00	32. Commit to transfer in TDNE high byte
130				B1	L	00	32. Commit to transfer in TDNE low byte (0: not, 1: committed)

Byte Offset	MB Reg Offset	Message	Message Offset			Raw (Base 16)	
131	64	33	32	B0	H	01	33. Test mode engine start only low byte (0: no load transfer, 1: load xfer, 2: disable test)
132				S	L	00	status
133	65			B2	H	01	34. Engine run test time (minutes) low byte (0-600 min)
134				B1	L	00	33. Test mode engine start only high byte
135	66	34	33	B0	H	00	34. Engine run test time (minutes) high byte
136				S	L	00	status
137	67			B2	H	00	35. Subnet pretransfer time (seconds) high byte
138				B1	L	03	35. Subnet pretransfer time (seconds) low byte (0-120 sec)
139	68	35	34	B0	H	01	36. Number of generators low byte
140				S	L	00	status
141	69			B2	H	03	37. 3P or 1P monitoring high byte
142				B1	L	00	36. Number of generators high byte
143	70	36	35	B0	H	00	37. 3P or 1P monitoring low byte (1 or 3)
144				S	L	00	status
145	71			B2	H	00	38. Subnet sequencing timer (seconds) high byte
146				B1	L	03	38. Subnet sequencing timer (seconds) low byte
147	72	37	36	B0	H	02	39. PT ratio low byte read only 2-500(:1)
148				S	L	00	status
149	73			B2	H	01	40. Closed transition on/off low byte (0: disabled, 1: enabled)
150				B1	L	00	39. PT ratio high byte
151	74	38	37	B0	H	00	40. Closed transition on/off high byte
152				S	L	00	status
153	75			B2	H	00	41. Closed transition phase angle difference high byte
154				B1	L	05	41. Closed transition phase angle difference low byte (0-10 deg)
155	76	39	38	B0	H	03	42. Closed transition frequency difference low byte (0.0 - 0.3 Hz)
156				S	L	00	status
157	77			B2	H	06	43. Closed transition voltage difference low byte (1-5%)
158				B1	L	00	42. Closed transition frequency difference high byte
159	78	40	39	B0	H	00	43. Closed transition voltage difference high byte
160				S	L	00	status
161	79			B2	H	00	44. In-phase transition on/off high byte
162				B1	L	00	44. In-phase transition on/off low byte
163	80	41	40	B0	H	02	45. In-phase transition phase angle difference low byte (0.0-60 deg)
164				S	L	00	status
165	81			B2	H	0A	46. In-phase transition frequency difference low byte
166				B1	L	00	45. In-phase transition phase angle difference high byte
167	82	42	41	B0	H	00	46. In-phase transition frequency difference high byte
168				S	L	00	status
169	83			B2	H	00	47. Maximum synchronization time (minutes) high byte
170				B1	L	05	47. Maximum synchronization time (minutes) low byte
171	84	43	42	B0	H	FD	Checksum (sum of previous 42 messages) low byte
172				S	L	00	status
173	85			B2	H	02	Complement of Checksum low byte
174				B1	L	07	Checksum (sum of previous 42 messages) high byte