SYNERGY SYSTEMS, LLC
Time proven products and support

Motorola ONCORE™ GPS Receiver Binary Communications

rev 28OCT03

by Randy Warner
Senior Applications Engineer, Synergy Systems, LLC

Introduction

Many times new users of the Motorola ONCORE™ series of GPS receivers have difficulty making the jump from using established GPS communications and display programs such as WinOncore12 and SynTAC to writing their own communications utilities that will communicate with the receiver through the serial port. The interface itself is really quite simple and straightforward, being no different than the interface one would design for any other embedded sensor that communicates with a host through a serial port.

The problem that catches most users is that when you are using these programs, the programs themselves are doing a lot of work in the background, work that needs to be tended to by the user (or the user's software) when communicating directly with the receivers through a custom designed console application or a microcontroller.

Other difficulties arise from deciphering the explanations of the binary protocol in the Motorola GPS User's Guides, which can be a LITTLE confusing, to say the least. One minute you might find yourself converting hex values generated in the native Motorola binary protocol to their ASCII equivalents, and the next you might be doing exactly the opposite. In the manual some messages are described in hex format, some in ASCII. Knowing which is which is extremely important.

On top of this, different receivers respond to different commands and protocols. For instance, positioning receivers generally respond to both the Motorola binary and NMEA commands, while timing receivers will only communicate in the native Motorola binary protocol. In addition, some of the commands and data strings used with 12 channel receivers such as the M12 and M12+ are different than those used with older receivers such as the VP, GT+, and UT+.

With that little preamble out of the way, let's just dive right in. I am first going to demonstrate how to enter commands in WinOncore12's <Msg> window, and then how you can construct the same commands for use in your own software. Remember, it's not difficult, it's just that it can be a little confusing until you see it done a couple of times. Although the commands I will be showing are for the M12+ receiver, the formatting rules are the same for any Motorola receiver operating in binary mode.

Motorola Binary Protocol

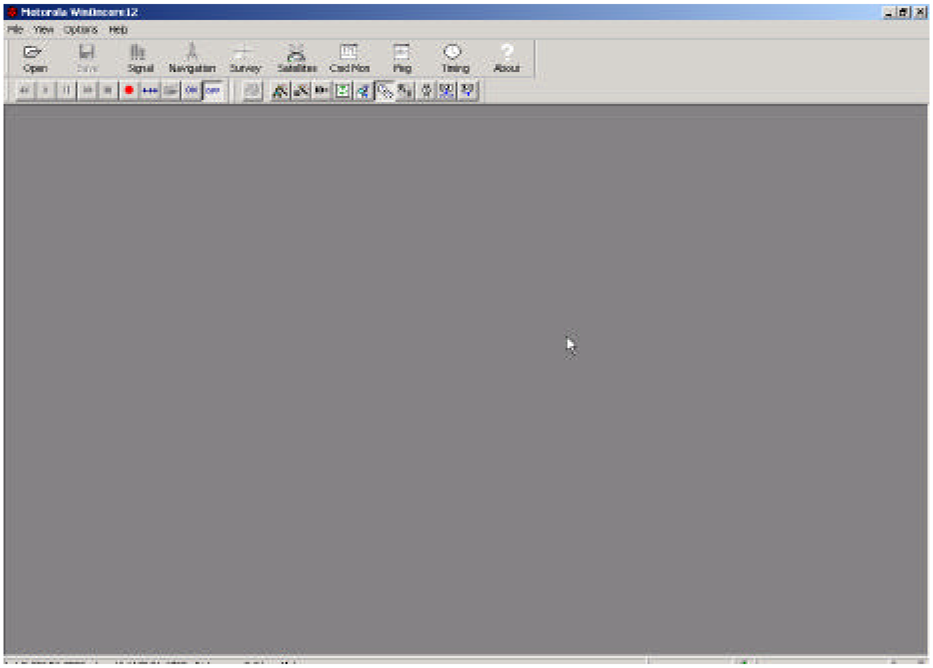
First of all, all binary communications with the Motorola receivers are conducted at 9600 baud, with 8 data bits, no parity, and one stop bit (standard 8/n/1 protocol). Also, generic terminal programs (Hyperterminal, etc.) are of limited value in communicating with the receivers in binary mode since they are primarily designed to send and receive the standard printable ASCII characters. They are not intended to handle all of the unprintable hex control characters that are transferred between the terminal and the ONCORE™ receiver when operating in binary mode without some sort of translation software. Many first time users are perplexed by the flood of “@@Hbo/%%_+~Nw*9..." strings being displayed by the terminal emulator when they first turn it on and try to monitor the communications. As I mentioned before, these “terminal emulator” programs are really designed to display normal printable ASCII characters. Your terminal program is simply displaying its interpretation of the ASCII equivalents of the hex characters being sent out by the receiver. Some of the characters are standard ASCII characters, which will display properly, while most of them translate into unprintable control characters. We need a little more intelligent program to properly display the data.

Communicating in Binary

Using the WinOncore12 software is certainly the quickest way to start out. The only thing one has to look out for is that older versions of WinOncore12 might not support some of the more recently added messages. Specifically, v1.0 of WinOncore12 does not support the timing related messages added in the M12+ Timing receiver. If you have the older version, simply go up to our website: www.synergy-gps.com and download the latest version (v1.2 as of this writing.)

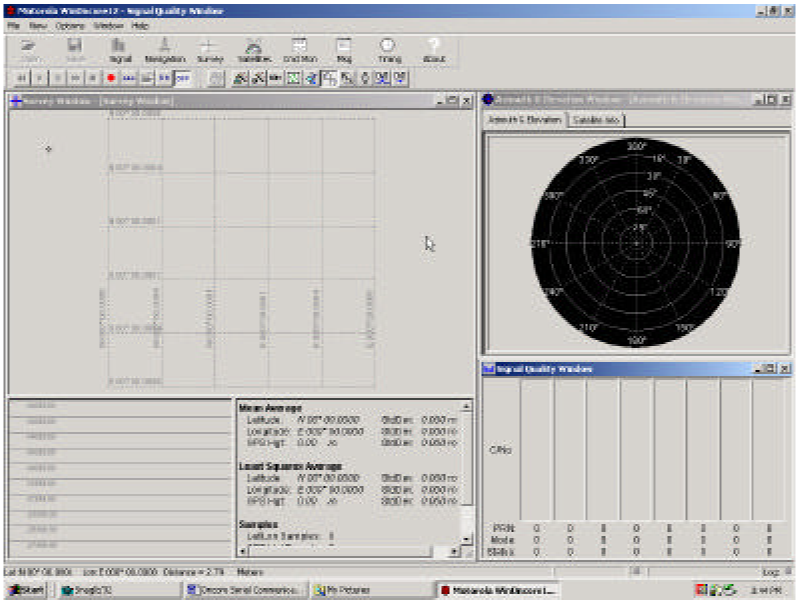
While you can use the startup "Wizard" to make initialization automatic, I am going to demonstrate initializing the receiver using individual binary commands entered into WinOncore12's <Msg> window. Once you have seen how these commands are formatted using WinOncore12 you should be able to format your own command strings for use with your own application's software.

Initialization Using WinOncore12



Before we can go much further, we need to get WinOncore set up. Shown above is the screen displayed the first time WinOncore is run after being installed. As you can see, it is not really ready to be used yet as none of the user windows are open. By clicking on the icons on the main

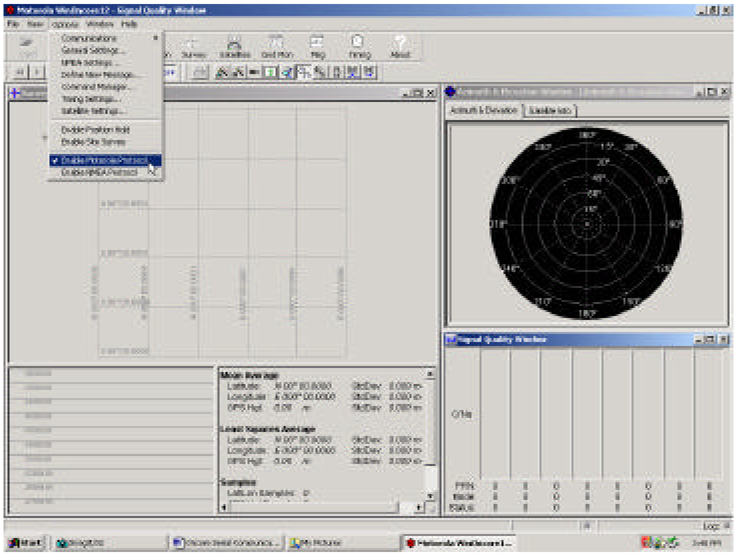
Toolbar I usually open the <Signal>, <Survey>, and <Satellites> windows as they display the information I usually find most useful. After pulling the windows around and resizing them the screen should look something like this:



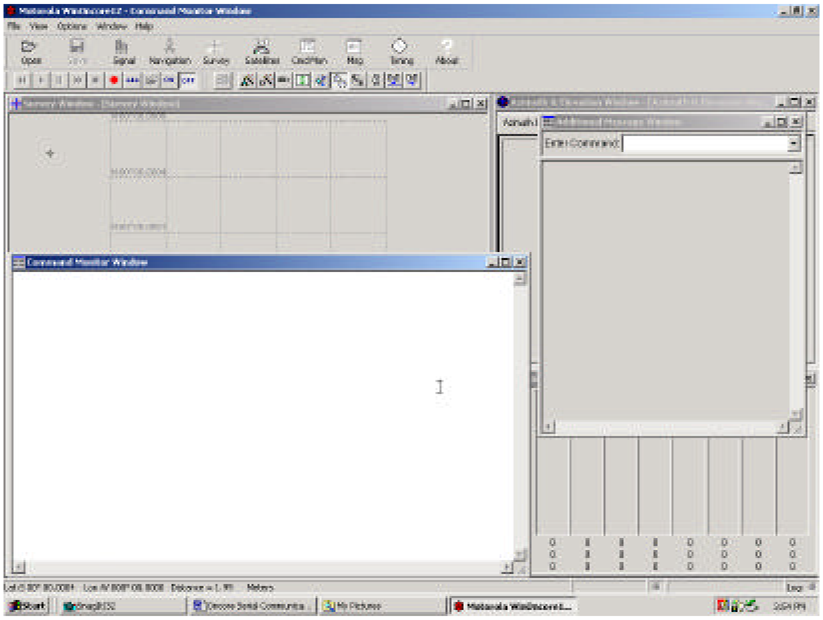
As you can see, I now have three windows open that can show average receiver position, satellite locations, and the C/No numbers for the satellites being tracked. Note that the <Survey> window displays running mean and least squares averages of the position reports. If you want to see un-averaged numbers you can open the <Navigation> window. This will display second by second positions along with other useful information about the current status of the receiver.

Now that we have a usable screen displayed we need to request that the receiver send out data for display. The simplest way to initialize the receiver is to do the following:

1. Click on <Options> and make sure that "Enable Motorola Protocol" is checked as shown below:

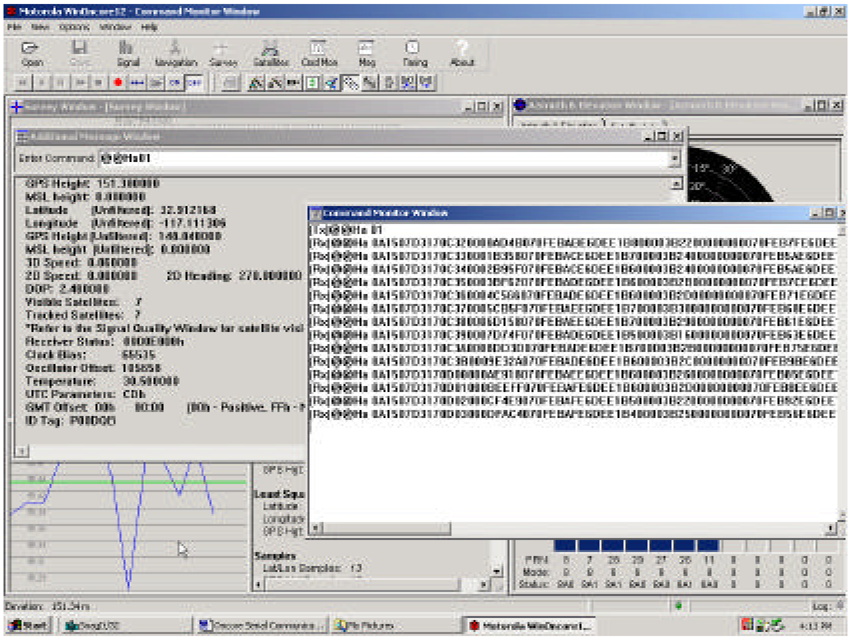


2. Open the <Msg> window so that we can enter a command to send to the receiver, and the <Cmd Mon> window so that we can watch the commands being sent to the receiver and the data coming back. After dragging the windows around a little, the screen should look something like this:



Motorola WinOncore12 - Command Monitor Window
File View Options Window Help
Open
Save
Signal
Navigation
Survey
Satellites
Cmd Mon
Msg
Timing
About
Survey Window - [Survey Window]
Command Monitor Window
Additional Message Window
Enter Command:
Start

3. Next we are going to send the @@Ha binary command which will direct the M12+ receiver to output the 12 channel Position/Status/Data message. In the <Additional Message Window> we type: ***@@Ha01<Enter>***. The receiver should start sending out @@Ha responses as a 1 Hz rate. Your screen should look something like this:



Motorola WinOncore12 - Command Monitor Window
File View Options Window Help
Open
Save
Signal
Navigation
Survey
Satellites
Cmd Mon
Msg
Timing
About
Additional Message Window
Command Monitor Window
Start

Enter Command: @@Ha01

```
GPS Height: 151.300000
MSL height: 0.000000
Latitude    (Unfiltered): 32.912168
Longitude   (Unfiltered): -117.111306
GPS Height (Unfiltered): 149.040000
MSL height (Unfiltered): 0.000000
3D Speed: 0.060000
2D Speed: 0.000000        2D Heading: 270.000000
DOP: 2.400000
Visible Satellites:    7
Tracked Satellites:    7
*Refer to the Signal Quality Window for satellite vis
Receiver Status:   0000E000h
Clock Bias:        65535
Oscillator Offset: 105658
Temperature:       30.500000
UTC Parameters:    C0h
GMT Offset: 00h    00:00    (00h - Positive, FFh - N
ID Tag: P000Q0
```

```
[Tx]@@Ha 01
```

Now that we have data flowing it is time to dissect what we have done to get this information from the receiver.

When we entered the ***@@Ha01<Enter>*** string into the 'Additional Message' window, WinOncore12 did not just send the ASCII characters "@" "@" "H" "a" "0" "1" to the receiver. In the first place, only the "@" "@" "H" "a" characters are ASCII. The "0" and "1" are actually the two nibbles of a hex "0x01". This switch in character definition is historical in nature and the explanation in the text of the User's Guide is not very clear. When you see a character string defined in this way, you need to treat the message identifier characters (@@Ha in this case) as ASCII, and anything following as hex.

Also, in the background WinOncore12 is calculating a checksum and appending Carriage Return/Linefeed characters to the data request. If you were to express the string entirely in ASCII characters it would look like:

'@' '@' 'H' 'a' '^A' '(' '^M' '^J'

where the caret (^) designates the <Control> key.

Not very pretty, but this is what is going on. Personally, when communicating with the receiver with a microcontroller I fund it much clearer to convert the entire string into hex. This would result in a command string of:

0x40 0x40 0x48 0x61 0x01 0x28 0x0d 0x0A

Now, suppose we would like a position update every 60 seconds. All we have to do is type in "@@Ha60"<Enter>, right? Of course not. Remember that when you are entering "@@Xx" commands that the command argument is in hex, therefore a value of 60_h will result in getting you a position update every 96 seconds. To get a response every 60 seconds the correct command is "@@Ha3C"<Enter>.

Command Mnemonics

The mnemonics can help a newcomer getting started with the ONCORE receivers remember the various commands. Let’s face it, “ps12” (the mnemonic for the 12 channel Position/Status message) is a little more intuitive than the actual “@@Ha” command header.

Of course, sending the receiver the "@@Ha" string doesn't do much good unless you request an update rate. Assuming we desire a position update every second, the full mnemonic is "ps12 1" <ENTER>. In fact, you can request a single response (polled mode) by entering “ps12 0" <ENTER>, all the way up to once every 255 seconds with the entry "ps12 255" <ENTER>. WinOncore12 will take care of checksums and make sure that the Carriage Return/Linefeed characters are appended to the string when you press ENTER just as before.

The thing to remember is that WinOncore12 is sending out the SAME EXACT hex string sent earlier with the '@@Ha01' command:

0x40 0x40 0x48 0x61 0x01 0x28 0x0d 0x0A

The only thing that has changed is how you formatted your request through WinOncore12.

Constructing Your Own Messages

Now that we have been through the basics of communicating with the receiver using WinOncore12, let's try writing our own commands and receiving data back from the receiver. Shown below is a little test program I wrote some time ago that shows how to initiate basic communications with the M12+. The target for this program is the Basic Stamp IISX, which has several unique serial communications macros that automate a lot of the port setup chores in the 'Constant Declaration' section, but no matter which processor you use the basic message formatting will be the same. I have added numbers to the lines that we will be discussing in detail. The comments in the code should explain everything else.

```
'                                   Ha COLD START TEST PROGRAM
'
'         This program initializes any Motorola M12/M12+ and starts the
'         @@Ha message at a 1 Hz rate. The program will allow the receiver
'         to keep running until a 3D fix is obtained as indicated by
'         bits 13, 14, and 15 in the Receiver Status word (byte 130 in the
'         Ha message) going high. It will then allow the receiver to continue
'         running for an additional 30 seconds to collect data for post processing,
'         default the receiver to restart the acquisition process, and repeat
'         the cycle until power is removed.
'
'                      Randy Warner - Synergy Systems LLC              05JUL02
'
'         constant declarations
'
N9600           con       240                 'set serial port for 9600 baud
N4800           con       500                 'set serial port for 4800 baud
GPSDATA         con       1                   'define GPS data port as Pin P1
GPSCMD          con       2                   'define GPS command port as Pin P2
C_R             con       $0D                 'Carriage Return character
L_F             con       $0A                 'Line Feed character

'         variable declarations

RX_FLAG         var       byte
COUNTER         var       byte

'         main program

(1)  serout GPSCMD,n4800,["$PMOTG,FOR,0",C_R,L_F]     'set to binary format in case receiver is in NMEA
pause 1000

restart:

RX_FLAG = 0                                                     'Initialize variable RX_FLAG
(2)  serout GPSCMD,n9600,[$40,$40,$43,$66,$25,C_R,L_F]          'Default Receiver
(3)  serout GPSCMD,n9600,[$40,$40,$41,$67,$0A,$2C,C_R,L_F]      'Set Mask to 10 Degrees
(4)  serout GPSCMD,n9600,[$40,$40,$48,$61,$01,$28,C_R,L_F]      'Request @@Ha string at 1Hz
'pause 1000

          loop:
(5)                 serin GPSDATA, n9600,[skip 129,RX_FLAG]
(6)                 IF RX_FLAG & %11100000 = %11100000 THEN collect_data      '3D fix bits high
                    pause 300
          goto loop

collect_data:

COUNTER = 0
          FOR COUNTER = 1 to 30
          pause 1000
          NEXT
goto restart
```

OK, let's dissect this little bit of code:

Line 1- directs the microcontroller to send out a NMEA string to tell the receiver to switch to the binary protocol. This is basically just a 'safety net' command in case the receiver you are using happens to be operating in the NMEA protocol. If the receiver is in NMEA, it will switch into binary. If the receiver is in binary already the command will simply be ignored.

Line 2 - sends the 'Default' command to the receiver. This command is used to turn off all binary messages that might be active so that the simple little microcontroller doesn't get inundated with a lot of data that it doesn't know how to deal with.

Line 3 - sets the receiver's Mask Angle to 10 degrees. This prevents the receiver from tracking satellites that are very close to the horizon that might cause position errors due to excessive ionospheric anomalies. Although not shown here, the receiver will echo back the command string so that you can be sure that the command was received and acted upon.

Line 4 - requests the @@Ha message from the receiver at a 1 Hz rate, just as we did earlier using WinOncore12.

Line 5 - Once we get the receiver to send out the @@Ha strings we drop into a loop. Since we want to look at the bit field in byte 130 of the @@Ha messages, Line 5 causes the microcontroller to skip over the first 129 bytes and then save byte130 to the variable RX_FLAG.

Line 6 - performs a bit-wise AND between RX_FLAG and the binary bit field '11100000'. If bits 5, 6, and 7 are high (with the right-most bit being bit 0) we know that the receiver has acquired enough satellites to compute a position. This causes the microcontroller to drop into the 'collect data' routine for 30 seconds and then start the whole process over again at 'restart'.

Rolling Your Own Code

Basically, if you got past the last part, you know how to do this. Whether you are working with an embedded controller in assembler, or coding in C++, all you have to do is output the hex string shown earlier through the serial port at 9600/N/8/1 and wait for the receiver to respond. Of course, unless you are designing a very simple interface or working with something like the Basic Stamp, additional lines of code will be required to:

Set up the serial port
Decide which message(s) to send
Decide on required command arguments (update rates or modes as applicable)
Calculate Checksum, and
AddCarriageReturn/LineFeed

Hey, nobody said it would be quick. But when you get right down to it, establishing communications with the ONCORE is the EASY part.