Xerxes Mount Control Protocol

Bruce Van Deventer,
Owner, Xerxes Scientific, LLC

Overview:

This white paper describes the design and requirements for the Xerxes Scientific DDR mount control protocol, which is a two way messaging protocol over Ethernet. The communications is (presently) between two computers, one is the mount computer which physically runs the mount, and the second is the user's computer, which also connects to the user's astronomy apps.

The communication takes place via Ethernet (UDP) datagrams of fixed length, sent at a nominally fixed rate on both ends. The design intent is for a high speed, lightweight approach which is inherently tolerant of lost or occasionally misordered (or weakly ordered) packets but for which commands and data are coherent. The scheme was intended to support customer-specific drivers for multiple platforms, with ASCOM being one specific interface protocol/customer platform.

Details

The design paradigm is for the mount and the mount driver to each blindly send data packets (datagrams) at a fixed rate, with each datagram containing all possible information that is needed by the other party. There are therefore two datagram types, the mount command datagram, and the mount status datagram. The mount command datagram contains all possible commands and all parameters in one single coherent record, and likewise, the mount status datagram contains one single coherent record of all data on the mount as a snapshot. Each datagram type contains rolling counters to establish that data is fresh, and the mount status datagram contains acknowledge flags for each command type, which are used to signal that the mount has received one or more commands in question. Mount commands are based on the ASCOM commands and mount status data is based mainly on ASCOM status data. An advantage of this scheme is that commonly used high data rate parameters from the mount, such as RA and Dec, are always as freshly available as possible. The cost of this is that the design protocol itself must manage messages that need acknowledgement, so this is done with command acknowledge flags in every mount status packet.

As an example, this is the sequence of events that take place when a slew command is issued from the ASCOM driver (or your driver) is as follows:

1. ASCOM driver is presented with an asynchronous slew request (or your driver wants to issue a slew request).
2. Driver populates the target dec, target RA fields in the command datagram with the requested values, and sets the slew command flag to true.

3. A periodic task (a timer event) in the ASCOM driver is triggered and sends the command datagram. A second task in the ASCOM driver waits for, reads, and processes input datagrams from the mount.
4. The mount software collects and stores the latest command from the mount, and sees the slew command change from false to true. The change of the command to active is the trigger to read the RA and Dec, calculate the motor drive slew info, start the slew, and set the slewing flag to true. It will set the slewing ACK flag to true in the mount status datagram. The mount will then slew to the coordinates with no further intervention.
5. When the ASCOM driver picks up that the slewing command ACK is true, it turns the slewing command request off in the command datagram, since it knows that the mount has received it and is acting on it.
6. When the mount sees the command removed, it removes the ACK for the command.

Note that in this scheme, an older datagram can be received that say slewing status is false immediately after the user program or ASCOM driver sends the slew request command, and that may fool the client into thinking that the slew is complete. The ASCOM driver has a workaround for this but user code which doesn't use the ASCOM driver should re-check that slewing is actually complete and not latch the status immediately after issuing the request.

Because the mount command datagram must contain all possible commands, it's necessary to have a prioritization scheme. In this design, prioritization of concurrent commands is performed in the mount. All active commands are examined and the highest priority command is serviced and acknowledged. The ASCOM driver or the user's driver is responsible to remove an active command once it's been acknowledged, and it is understood that the command may be repeated for a few cycles before it is removed. The mount software only acts on the "rising edge" of the command (the transition from not active to active), then the command is launched, and the ack flag in the datagram is set for that command. When the driver program sees the ack command, it knows that the command has been received and it removes the command.

Commands in the mount are primarily intended to execute asynchronously, and if synchronous commands (in ASCOM-speak) are required, it's the mount driver's responsibility to wait on the mount reply datagram contents to see that the action was completed if it is the type of command that takes time.

Xerxes Mount Command Datagram

The table below shows the Xerxes Mount command 82 byte datagram format. This is to be sent to the mount at a 50ms (20Hz) period using a UDP connection.

bytes	***Field***	***type***	***bytes***	***Description***
0-7	CmdHeader	Uint64	8	command header & permission code (0x5555aaaa)

8 to 15	rolling counter	long int	8	rolling counter of commands sent to mount
16 to 23	TargetDec	double	8	target declination
24 to 31	TargetRA	double	8	target RA
32 to 39	MoveAxisRateDec	double	8	move axis rate in dec in degrees per second
40 to 47	MoveAxisRateRA	double	8	move axis rate for RA in degrees per second
48 to 55	DecFineRate	long int	8	fine rate, driver converts to tics/sec x 10
56 to 63	RAFineRate	long int	8	fine rate, driver converts to tics/sec x 10
64 to 67	PulseGuideRADuration	signed int	4	RA pulse guide duration. If zero, don't guide. Otherwise, guide, sign indicates direction, neg is east
68 to 71	PulseGuideDecDuration	signed int	4	Dec pulse Guide duration / direction. If zero, don't guide. Otherwise, sign indicates direction and duration
72	tracking rate	byte	1	enumerated 0 thru 3 for tracking rate (not used)
73	AbortSlewCmd	boolean	1	true to abort the slew
74	FindHomeCmd	boolean	1	true to move to home position (not used)
75	MoveAxiSRA	boolean	1	true to move RA axis, note that both can be true at once so there is a composite command, moveaxis, if either is true
76	MoveAxisDec	boolean	1	true to move dec axis
77	ParkCmd	boolean	1	true to park the mount (disable drives)
78	PulseGuideCmd	boolean	1	true to use values in PulseGuideDuration on both axes. Sets timer in mount SW, which runs to expiration.
79	SlewToTargetCmd	boolean	1	true means use target ra, dec pair as the value to slew to. Ignore if still slewing.
80	SyncToTargetCmd	boolean	1	true means change scope RA, dec reported values to the sync value.
81	spare	boolean	1	spare

CmdHeader is &H5555AAAA

Note that there is no endian flag for this data so far; it is based on Windows BitConverter functions which operate on byte arrays and assumes the standard Windows.NET endian-ness.

RollingCounter is to be incremented by the sender for each message sent, currently only used for diagnostic purposes.

TargetDec and ***TargetRA*** are Doubles in the usual ASCOM format for RA and dec (verify RA is 0 to 24 and dec is -90 to 90).

MoveAxisRateDec and ***MoveAxisRateRA*** are in degrees/second and are used for satellite tracking.

DecFineRate and ***RAFineRate*** are in encoder tics/sec x 10 and are tracking rate fine adjusters. On this mount, there are 2^26 encoder tics per revolution, and speed is expressed in tics/sec x 10. So for example, sidereal rate is roughly 7789. (778.9 tics/sec x 10).

PulseGuideRADuration and ***PulseGuideDecDuration*** are the pulse guider durations in milliseconds, as signed values. The sign is used to distinguish E/W vs N/S.

TrackingRate is the ASCOM enumerated tracking rate, currently only as sidereal.

AbortSlewCmd is a byte long Boolean flag (0=false, true = FF). It will slow and then stop drive but keep the drive active. Note that there are other options such as shutting off the drive without slowing it down or doing a rapid (current limited) stop and then shut off.

FindHomeCmd will in the future start a slew to the home position, without cord wrap.

MoveAxisRA and ***MoveAxisDec*** are commands that use the moveaxis rates to perform high speed satellite tracking. Those rates are supposed to be additive to the tracking rates.

ParkCmd puts the telescope in park mode, which depowers (disable) the servo drives. Park does not currently slew the mount prior to parking, it disables the drives wherever the mount is.

FYI-the absolute encoders are offset from this position based on how the mount was assembled. The home position is used internally as part of the cable wrap protection logic.

PulseGuideCmd is used to start pulse guiding. The mount starts an internal timer and runs the mount at a fixed altered rate on the specified drive until the timer expires.

SlewToTargetCmd uses the targetDec and targetRA values and starts a slew to the target. The mount will report a slewing status bit as true while it's still slewing.

SyncToTarget cmd tells the mount to take the current targetDec and targetRA values in this datagram and sync to them.

The mount will also produce its own status datagram autonomously and at a 20Hz rate:

The Mount Status Datagram format is here (Appendix A contains the code to pack):

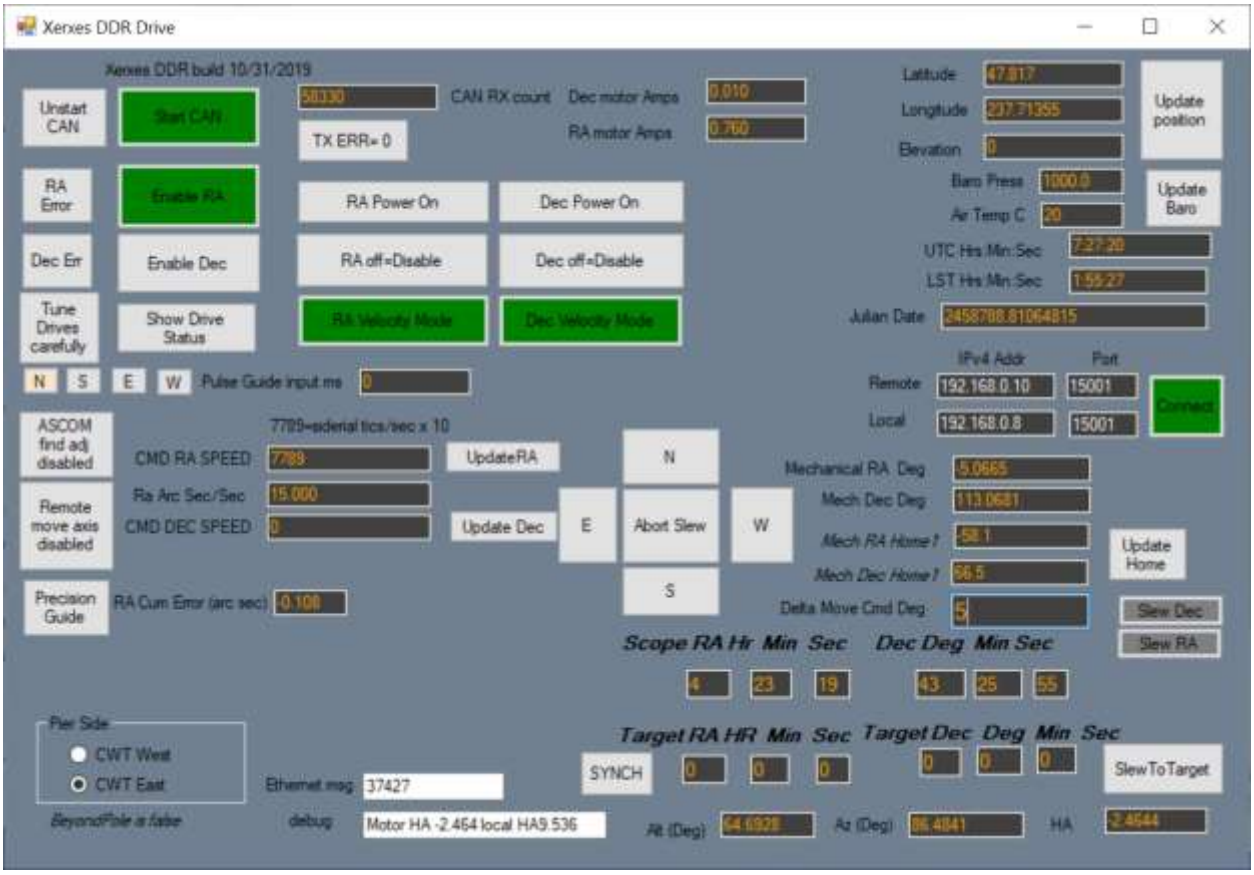
byte #	size	ASCOM Property Name	Format	Description
0 to 7	8	Description	String	"XERXESxx" header
8 to 15	8	Altitude	Double	ASCOM compatible altitude
16 to 23	8	Azimuth	Double	ASCOM compatible elevation
24 to 31	8	Declination	Double	Declination -90 to 90.
32 to 39		DeclinationRate	Long	Servo drive rate in guiding in tics per second x 10 (2^26 tics per revolution)
40 to 47	8	RightAsension	Double	RA in 0 to 24.0 ASCOM format
48 to 55	8	RightAscensionrate	Double	Servo drive rate in guiding in tics per second x 10 (2^26 tics per revolution). Sidereal rate is 7789.
56 to 63	8	SiderealTime	Double	LST as reported by the clock in the mount.
64 to 71	8	SiteElevation	Double	ASCOM SiteInfo.Elevation property
72 to 79	8	SiteLatitude	Double	ASCOM SiteInfo.Latitude, read only

80 to 87	8	SiteLongitude	Double	ASCOM SiteInfo.Longitude from mount, read only
88 to 95	8	TargetDeclination	Double	Target declination used for slews or sync. The ASCOM driver must always populate this when using any sync or slew command. Epoch is Jnow (for now, but probably we will change this).
96 to 103	8	TargetRightAscenscion	Double	Target RA used for slews or sync. The ASCOM driver must always populate this when using any sync or slew command. Epoch is Jnow (for now).
104 to 111	8	Rolling Counter	N/A	Rolling tic counter from master event timer in mount, which should increment once per outbound message.
112 to 119	8	GuideRateDeclination	Double	Encoder tics/sec x 10, currently hard-coded to 3500 which is roughly half sidereal rate. Used in pulse guiding.
120 to 127	8	GuideRateRA	Double	Encoder tics/sec x 10, currently hard-coded to 3500 which is roughly half sidereal rate. Used in pulse guiding.
128 to 131	4	DecRawCurrent	N/A	Servo drive current in 10mA units. Typical drive current is around 0.5 to 2 amps.
132 to 135	4	RARawCurrent	N/A	Servo drive current in 10mA units. Typical drive current is around 0.5 to 2 amps.
136	1	AtHome	boolean	0=False, FF= true, if mount is at “Home” (not implemented yet)
137	1	AtPark	boolean	0=False, FF= true, if mount is at “Park”
138	1	Connected	boolean	Currently hard coded to true.
139	1	DoesRefraction	boolean	Currently false, but will be used in the future.
140	1	EquatorialSystem	enumerated	0 = Jnow only. Uses the ASCOM meaning.
141	1	IsPulseGuiding	boolean	Used when either pulse guiding timer is active in RA or Dec
142	1	SideOfPeier	enumerated	This becomes beyondThePole, is implemented.
143	1	Slewing	boolean	True while slewing.
144	1	Tracking	boolean	Always true for now
145	1	TrackingRate	enumerated	0 = sidereal, hard coded.
146	1	**AckMoveAxisDec**	boolean	Ack flag for a MoveAxis command in De. Slewing will be true for a MoveAxis Commsnd.
147	1	**AckMoveAxisRA**	boolean	Ack flag for a MoveAxis command in RA. Slewing will be true during a MoveAxis command.
148	1	UTCDateTime	future	
149	1	UTCDateTime	future	
150	1	UTCDateTime	future	
151	1	RADrivestatus1	future implementation	

152	1	RADriveStatus2	future implementation	
153	1	DecDriveStatus1	future implementation	
154	1	DecDriveStatus2	future implementation	
155	1	AckAbortCmd	boolean	Ack for an abort slew command.
156	1	AckSyncCmd	boolean	Ack for the sync coordinates command
157	1	AckPulseGuideCmd	boolean	Ack for the pulse guide command
158	1	AckSlewCmd	boolean	Ack for the slewing command
159	1			spare

Mount Control Design Theory of Operation:

The mount control code itself is implemented as a Windows .NET executable, with all the capabilities provided by the .NET framework. The mount user is presented with a GUI (shown below), which is used to start the mount, monitor operation, and "manually" operate the mount. This software has two main control features: a timer loop running at 50ms and a UDP listener thread. The 50ms thread implements all CAN and CANopen protocol / EtherCAT commands which communicate with the servo drives. This thread controls all modes of operation and provides all message updates, including updating and transmitting the outbound mount status datagram. The listener thread simply receives datagrams from the specified port and makes them available to the control program.



<figure shows an early version of the display>

The user types in the IPV4 address of the computer with the ASCOM driver and the port that will be selected by the ASCOM connect menu pick, as well as the local IPV4 address (which is pre-populated) and a port field which defaults to 15001. Clicking Connect simply starts the UDP listener task and starts sending datagrams from the program. The "debug" line currently displays the number of datagrams received but that will go to a dedicated field.

In summary, the mount simply sends datagrams every 50ms (approximately, as the Windows forms timer is imprecise) and also listens for commands. There is by design no synchronization between the listener task and the timer loop; the inbound message should be thought of as a "mailbox" and the mount just goes in and processes the latest message and anything previous is ignored. When the mount has data to send, it sends a message containing everything. The inbound message must contain all data necessary to launch any mount command, and the outbound message must contain all data needed by the ASCOM driver to understand the mount state.

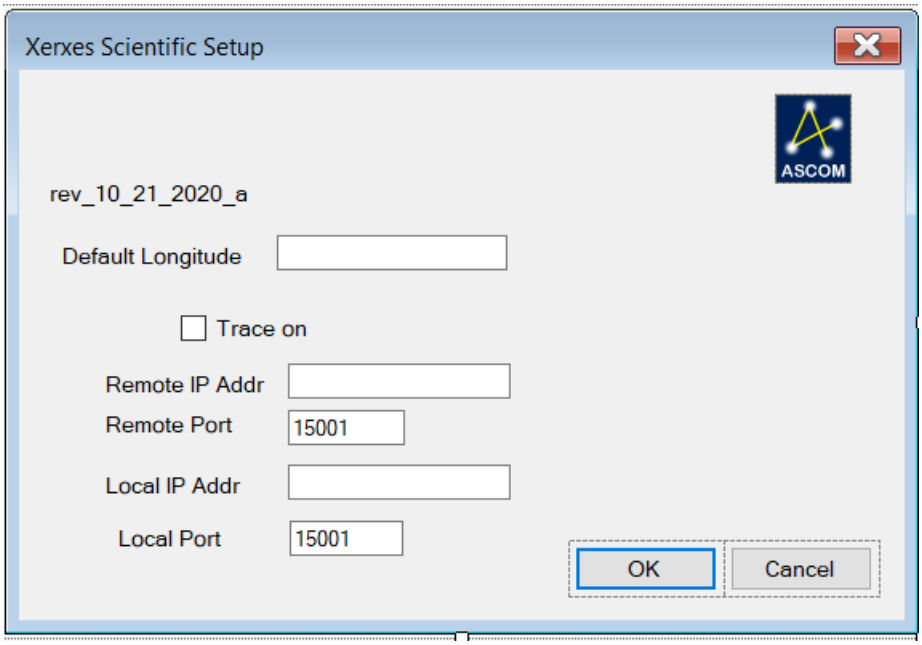
HOME POSITION Important Note!

The Renishaw encoders are configured to provide full wrap capability, that is, it can report numbers from -360 degrees to +360 degrees (and beyond). However, the encoder physically only has one 360 degree scale, which has a zero position. If you start up the drive with the read head in the 0 to 180 half-circle, the drive will always see the start up position as a positive number. If you start the drive at beyond 180 degrees, the Renishaw encoder will report that as a negative angle, 0 to -180 degrees. Now, if in setting the home position you had powered up the mount at say 170 degrees but move over to 190 and then try to set that as a home position, when you re-apply power, the drive will think that is -170 degrees. In other words, wrap data is not preserved between power cycles but wrapping does occur within the same power cycle. The control logic in the program handles 170 and -190 degrees the same of course.

ASCOM Driver Theory of Operation

The driver is written in VB.NET and uses the standard ASCOM (32 bit COM) Telescope driver template. Instead of selecting a COM (serial) port and processing and sending a RS-232 command, the driver also has a timer (a System.Timers.Timer, which is different from the Forms timer) and a UDP listener thread. The timer runs every 50ms and transmits the current ASCOM command datagram, regardless of whether it has changed. The timer thread also looks at the most recently received datagram's ACK bytes and uses these to clear any commands that are active.

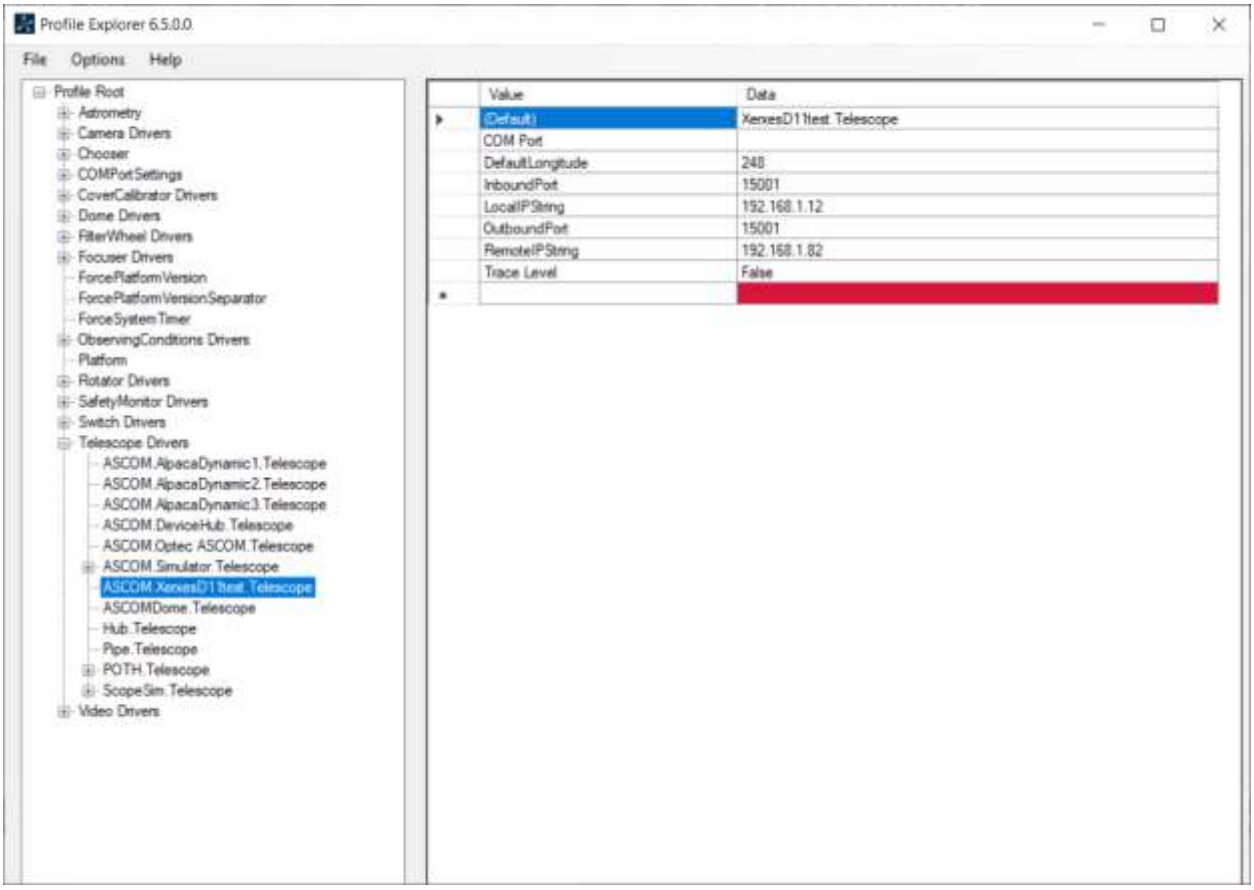
The "normal" ASCOM properties and methods operate on the receive data packet to read data from the mount and the command data packet to send commands and data to the mount. They do not directly send or receive data, they simply read or populate fields in the messages, which the event threads actually send and receive.



ASCOM D11Test setup dialog window. The default longitude must be populated but is not used.

Remote and local IP address cannot be the same (must be on different machines).

ASCOM profile explorer screen will look like this (note that COM port is not used):


Profile Explorer 6.5.0.0
File
Options
Help
Profile Root
Astrometry
Camera Drivers
Chooser
COMPortSettings
CoverCalibrator Drivers
Dome Drivers
FilterWheel Drivers
Focuser Drivers
ForcePlatformVersion
ForcePlatformVersionSeparator
ForceSystemTimer
ObservingConditions Drivers
Platform
Rotator Drivers
SafetyMonitor Drivers
Switch Drivers
Telescope Drivers
ASCOM.AlpacaDynamic1.Telescope
ASCOM.AlpacaDynamic2.Telescope
ASCOM.AlpacaDynamic3.Telescope
ASCOM.DeviceHub.Telescope
ASCOM.Optec ASCOM.Telescope
ASCOM.Simulator.Telescope
ASCOM.XerxesD11test.Telescope
ASCOMDome.Telescope
Hub.Telescope
Pipe.Telescope
POTH.Telescope
ScopeSim.Telescope
Video Drivers
Value
Data
(Default)
XerxesD11test Telescope
COM Port
DefaultLongitude
248
InboundPort
15001
LocalIPString
192.168.1.12
OutboundPort
15001
RemoteIPString
192.168.1.82
Trace Level
False

Appendix A;

Appendix A:

6/25/2019 Mount Status Datagram packing code:

```
    Private Sub PackDatagram()
        ' packs globals into a byte array and sends it
        ' for now it is just a test

        Dim ByteArray(8) As Byte '' array of length 8
        Dim MidByteArray(4) As Byte ' array of length 4
        Dim BigByteArray(160) As Byte ' big buffer of 160 bytes
        Dim k As Integer
        Dim temp_double As Double
        Dim temp_int32 As Int32
        Try
            BigByteArray(0) = Convert.ToByte(88) ' lead off with our header for our packet type =XERXESxx
            BigByteArray(1) = Convert.ToByte(69)
            BigByteArray(2) = Convert.ToByte(82)
            BigByteArray(3) = Convert.ToByte(88)
            BigByteArray(4) = Convert.ToByte(69)
            BigByteArray(5) = Convert.ToByte(83)
            BigByteArray(6) = Convert.ToByte(120)
            BigByteArray(7) = Convert.ToByte(120)
            ByteArray = BitConverter.GetBytes(Alt) ' pack altitude as double
            For k = 0 To 7
                BigByteArray(8 + k) = ByteArray(k)
            Next
            ByteArray = BitConverter.GetBytes(Az) ' pack azimuth as Double
            For k = 0 To 7
                BigByteArray(16 + k) = ByteArray(k)
            Next
            ByteArray = BitConverter.GetBytes(CurrentDec) 'pack RA as Double
            For k = 0 To 7
                BigByteArray(24 + k) = ByteArray(k)
            Next
            ByteArray = BitConverter.GetBytes(CurrentDecSpeed) ' pack Dec tracking speed as long int
            For k = 0 To 7
                BigByteArray(32 + k) = ByteArray(k)
            Next
            ByteArray = BitConverter.GetBytes(CurrentRA) ' pack current RA as Double
            For k = 0 To 7
```

```
    BigByteArray(40 + k) = ByteArray(k)
Next
ByteArray = BitConverter.GetBytes(CurrentRASpeed) ' pack current servo drive RA speed as long tics/sec x 10
For k = 0 To 7
    BigByteArray(48 + k) = ByteArray(k)
Next
ByteArray = BitConverter.GetBytes(CurLST) ' Pack current sidereal time as double
For k = 0 To 7
    BigByteArray(56 + k) = ByteArray(k)
Next
temp_double = MySiteInfo.Height
ByteArray = BitConverter.GetBytes(temp_double) ' observatory altitude in meters as double
For k = 0 To 7
    BigByteArray(64 + k) = ByteArray(k)
Next
temp_double = MySiteInfo.Latitude ' observatory latitude in degrees per ASCOM standard, as double
ByteArray = BitConverter.GetBytes(temp_double)
For k = 0 To 7
    BigByteArray(72 + k) = ByteArray(k)
Next
temp_double = MySiteInfo.Longitude ' observatory longitude in degrees per ASCOM standard, as double
ByteArray = BitConverter.GetBytes(temp_double)
For k = 0 To 7
    BigByteArray(80 + k) = ByteArray(k)
Next
ByteArray = BitConverter.GetBytes(TargetDec) ' slewing target declination as double
For k = 0 To 7
    BigByteArray(88 + k) = ByteArray(k)
Next
ByteArray = BitConverter.GetBytes(TargetRA) ' slewing target RA as double
For k = 0 To 7
    BigByteArray(96 + k) = ByteArray(k)
Next
' non-ASCOM master loop counter
ByteArray = BitConverter.GetBytes(MasterTicCount) ' masterticcount as double
For k = 0 To 7
    BigByteArray(104 + k) = ByteArray(k)
Next
' guide rates
ByteArray = BitConverter.GetBytes(GuideRateDec) ' Dec Guide rate in tics/sec x 10
For k = 0 To 7
    BigByteArray(112 + k) = ByteArray(k)
```

```
Next
ByteArray = BitConverter.GetBytes(GuideRateRA) ' RA Guide rate in tics/sec x 10, usually about 1/2 of sidereal
For k = 0 To 7
    BigByteArray(120 + k) = ByteArray(k)
Next
temp_int32 = DecRawCurrent
MidByteArray = BitConverter.GetBytes(temp_int32) ' Dec current as 10mA units
For k = 0 To 3
    BigByteArray(128 + k) = MidByteArray(k)
Next
temp_int32 = RARawCurrent
MidByteArray = BitConverter.GetBytes(temp_int32) ' RA current as 10mA units
For k = 0 To 3
    BigByteArray(132 + k) = MidByteArray(k)
Next
' now pack booleans
BigByteArray(136) = Convert.ToByte(0) ' At Home
BigByteArray(137) = Convert.ToByte(0) ' at park
BigByteArray(138) = Convert.ToByte(&HFF) ' true for Connected
BigByteArray(139) = Convert.ToByte(0) ' does refraction
BigByteArray(140) = 0 '   Equatorial System = Jnow
If (IsRAPulseGuiding Or IsDecPulseGuiding) Then
    BigByteArray(141) = Convert.ToByte(&HFF)
Else
    BigByteArray(141) = 0
End If
If BeyondPole Then ' beyondpole is used for the true value of side of pier in ASCOM
    BigByteArray(142) = Convert.ToByte(&HFF)
Else
    BigByteArray(142) = 0
End If
' now cover both axes for slewing, also set tracking which is 140 one as the opposite of the other
If (RASlewInProgress) Or (DecSlewInProgress) Then
    BigByteArray(143) = Convert.ToByte(&HFF)
    BigByteArray(144) = 0
Else
    BigByteArray(143) = 0
    BigByteArray(144) = Convert.ToByte(&HFF)
End If
' we treat tracking as the opposite of slewing.
BigByteArray(145) = 0 ' tracking rate is sidereal
```

```
        ' now we are done packing it, now send it
        If AckMoveAxisDecCmd Then
            BigByteArray(146) = Convert.ToByte(&HFF)
        Else
            BigByteArray(146) = 0
        End If
        If AckMoveAxisRACmd Then
            BigByteArray(147) = Convert.ToByte(&HFF)
        Else
            BigByteArray(147) = 0
        End If
        If AckAbortCmd Then
            BigByteArray(155) = Convert.ToByte(&HFF)
        Else
            BigByteArray(155) = 0
        End If
        If AckSyncCmd Then
            BigByteArray(156) = Convert.ToByte(&HFF)
        Else
            BigByteArray(156) = 0
        End If
        If AckPulseGuideCmd Then
            BigByteArray(157) = Convert.ToByte(&HFF)
        Else
            BigByteArray(157) = 0
        End If
        If AckSlewCmd Then
            BigByteArray(158) = Convert.ToByte(&HFF)
        Else
            BigByteArray(158) = 0
        End If
        SendDatagram(BigByteArray, 160) ' send 160 byte message
    Catch ex As Exception
        MsgBox("error on package datagram " & ex.Message)
    End Try
End Sub
```