VBScript Starter Kit: Simple automation scripting for *TheSkyX*™

Rick McAlister, Rev 2.1, February 2019

Overview

Do you want to automate *TheSkyX*™ with scripts but think you need to be a "programmer"? If you can do three things: 1) create and edit a file on a computer, 2) use algebra, and 3) write a list of steps, then you can do automation scripting. This primer gets you started with the bare necessities for writing and running a Visual Basic Script to automate simple *TheSkyX*™ functions. Within a just a few minutes, you will have built and run your first Visual Basic Scripting program.

This tutorial is divided into three parts. The first part explains the software tools you'll need and how to use them. The second part covers the basics of a VBS script that automates TSX. The third part presents a number of VBScripts and explains what each does and how they work, line by line.

Once you've worked your way this that you should be well on your way to writing your own VBScript automation scripts. You should be aware that, in the pursuit of simplicity, I have chosen but one way to write the scripts that follow. If you want to get more sophisticated, then there are good introductory books on writing more complicated VBS statements, operations and control structures. Go buy one. Read it. Have a blast. But first…

Part I: The Tools

We're going to need just three tools to automate TSX with VBScript:

1. *ScriptTheSkyX*™ documentation: the catalog of TSX's library of functions.
2. Notepad: An editor to open, edit, and saves a VBS program.
3. Windows Scripting Host: The application that actually runs your Visual Basic Scripting script.

1. *ScriptTheSkyX*™*:* Understanding the TSX automation library

Not all the functionality that is available to a TSX user through a mouse click can be scripted. The on-line document, *ScriptTheSkyX*™ is essentially a listing of all those functions that a user can automate, and how to access them. The reference is located online at http://www.bisque.com/scriptTheSkyX. On occasion, Software Bisque will add to these functions so it's good to check occasionally for updates. But for now, bring it up, page through it a bit, then come back here after you are thoroughly overwhelmed, but have an idea of what you're looking at.

Modern computer coding is based on something called "Object Orientated Programming". For our purposes, you skip past all that complicated structure and terminology. A deep understanding of OOP is practically useless for basic scripting. For us, what matters is that the automation functions of TSX are divided into groupings called "Classes". Each of these groups are associated with some device or functionality within TSX.

So, starting at the top level in *ScriptTheSkyX*™, select the "Classes" tab and look at the page that comes up. This is the list of class groupings for TSX functions. For instance, there is a function grouping named "sk6RASCOMTele" which contains all the mount control stuff. There is a function group called "sky6Dome" which contains all the dome control stuff. There is a function grouping named "sky6StarChart" which contains all the screen chart information and controls. And so on.

TSX automation contains a slew of these classes. Fortunately, you'll probably need just few for basic scripting purposes, mostly the hardware control classes (e.g. dome, mount, camera, imaging). *ScriptTheSkyX*™ does not provide much in the way of description for each class in the list other than the clue in the choice of names. If you're looking for something special then you may have to do some hunting down into the class documentation itself, but you'll get it figured out.

Now select the "ccdsoftCamera" class (the word "Camera" may give you a clue about what it contains, but not completely). This class is the grouping of functions associated with controlling the main camera, any guider camera, filter wheel, focuser and/or rotator. Page down through the definitions. Note that there are some "Public Types" (with "enum" on the left side), some "Slots" and some "Properties". Each of these serves a different purpose in using the TSX functionality and you'll generally see all of these types them within each class grouping. For the purposes of VBScripting, the difference between a Property and a Slot is that a Property generally reads or writes some data item and a Slot

usually causes something to happen. This is similar to the difference between a numeric entry and a command button in a TSX Window. For instance, in this ccdsoftCamera class, the Property "ExposureTime" writes (or reads) the exposure time for acquisition of an image. The Slot "TakeImage()" is used to cause the image (for that exposure time) to be acquired. Sometimes a Slot is also called "Method" or "Sub" or "Function". It all works.

The third type you'll see is prefaced with "enum" which means enumeration. These are important because they indicate successive number values for specific properties. For instance, if you look at the property "Frame" in the ccdsoftCamera class, you will see that this property sets the type of frame the camera will take when imaging. *ScriptTheSkyX*™ isn't explicit here, but it implies that the range of values that the Property can take are defined in the ccdsoftCamera class enumeration ccdsoftImageFrame. When you look to that enumeration, you'll see "cdLight", "cdBias", "cdDark", and "cdFlat", in that order.

Enumerations normally start at zero and increment by one for each successive type. If the enumeration starts at a different value, then *ScriptTheSkyX*™ will probably indicate it. For instance, look closely at the emumeration for ccdsoftImageFrame. The first entry, the "cdLight" frame type, equals a value of 1. So, a "cdBias" frame type will have a value of 2, a "cdDark" frame type will have a value of 3, and a "cdFlat" frame type will have a value of 4. The meanings of values that other properties can take are figured out in the same way throughout *ScriptTheSkyX*™.

You'll see some functions that seem obvious as to what they are for (e.g. Abort) and some that may not look so obvious (e.g. SetPropDbl). Just look around a bit, then we'll work on how to use them in a script.

2. *Notepad:* Opening, editing, saving a script

Using the Windows File Explorer, create a new directory in your Documents folder and rename it "VBScripts".

Use the start menu to open the Notepad application. You'll find it in Windows Accessories. Pin it to Start or create a short cut to your desktop – you'll be using it a lot.

Enter the following two lines with this editor, without leading blanks, exactly:

```
testname = "I made this myself"
msgbox (testname)
```

Then, in the File menu, use the "Save As" command. Navigate to the new directory you created, "Documents\VBScripts". Set the file type to "All files". Enter the file name "firsttry.vbs" and "Save". Then close the Notepad editor.

3. Windows Scripting Host -- Running a VBS script

Windows Scripting Host is normally included with every Windows operating system installation and will be the default application (double left click) for a *.vbs file, but that can be changed. This is the application that actually runs your script for you and fields any errors that may occur. It reads your script line by line and provides all the software connections to the operating system and other applications that your script may call on. It does all the heavy lifting. Most of the time you'll never even know it's there, any more than you know what's going on behind the scenes with Internet Explorer. It's one of the things that makes VBS so simple. Now let's get back to our first script.

Open Windows Explorer and locate your file "firsttry.vbs". Right click to open the file, then open with "Microsoft Windows-based Scripting Host". (Subsequently, when we reopen the file to edit with Notepad, do the same thing, but pick "Notepad" instead.) If you've done everything right, then a small window should open containing the phrase "I made this myself", and an "OK" button. Hit the "OK" and you have successfully completed your first VBScript.

Good news: the two statements you wrote cover much of what you will need to write simple control scripts for TSX. The first statement creates a variable name, much like algebra, that can be assigned values that the computer will understand. In this case, the statement assigns an ordered set of characters called a "string". The second statement calls another function, this time a built-in Windows function, that writes out the contents of any variable it is given, in this case, the

contents our string variable “testname”. Functions inside of TSX get handled essentially the same way. The function “msgbox” is not necessary for writing TSX scripts, but it can help a lot when debugging.

Part II: Writing and Testing VBScript using TSX functions

Now, let’s try one that uses a TSX function. Open your “firstlight.vbs” with the Notepad editor again. Clear the contents and enter the following statements.

```
set tsxo = CreateObject("TheSkyX.Application")
versionnumber = tsxo.version
msgbox (“TSX Version: “ & versionnumber)
```

Take time to study a couple of statements in this script. The “set” statement is used to establish a symbolic name such that the computer can set up and run a connection between your script and a particular class within a particular application. So, the “TheSkyX” part of the string means to use the TSX automation functions rather than, say, Excel’s. Then there’s the period (a.k.a. dot) and the “Application” part of the string. This means that “tsxo” will thereafter refer to the “Application” group (a.k.a. “class”) of slots and properties inside TSX.

So, right now – go look up the “Application” class in *ScriptTheSkyX*™ to see what functions are available to you in this TSX class, and what they do. Be aware that, before accessing each and every TSX class, you must do this “CreateObject” name assignment and connection. From then on out, the name that you have chosen will preface every slot or property in that automation class.

So in our little script, the property “tsxo.version” is how you access the “Version” property in the TSX “Application” class. More on that in a bit. The “&” tells the script to tack on the string “TSX Version” in front of the contents of that property (which we’ve placed in the “versionnumber” variable. By the way, if you don’t like “tsxo” for the class connection, you could substitute any odd name you wanted like “donkey” or “lastchanceforgas” or whatever. It’s all the same. I like to make up short names because they are easier to type. We’re ready to move to the next step.

TIP: If TSX is not running at the time the first “Create Object” call in your script is hit, then Windows Scripting Host will launch TSX for you. So, you can either launch TSX before running your script, or let the script to launch it for you. It doesn’t make much difference either way. However, if you write some script lines to specifically launch TSX (yes, you can do this), but it is already running, then the Windows Scripting Host will try to launch a second copy – not a good thing. The best method, then, is to let the Windows Scripting Host handle it for you when you create your first TSX object.

Save the file, close Notepad and run your new script just like last time. As the script completes, a window should open and display your version number. Now you’ve built and run your second Visual Basic Program and first TSX script. Congrats. No more excuses. You can no longer say that you’re not a programmer.

Let’s write a script to take an image. What we’re going to do is

1) set up a connection to the TSX camera class,
2) connect TSX to the camera,
3) set the exposure length to 10 seconds,
4) set the frame type to a light frame,
5) set the reduction type to Autodark, 6) turn AutoSave on, 7) take the image.

The VBScript statements look like this:

```
set tsxo = CreateObject("TheSkyX.ccdsoftCamera")
status = tsxo.Connect()
tsxo.ExposureTime = 10
tsxo.Frame = 1
tsxo.ImageReduction = 1
tsxo.AutoSaveOn = 1
status = tsxo.TakeImage()
msgbox("Image Done")
```

Note that I've used the variable "status" to pick up any error message out of each call, but I'm not really using it for anything. Now read the following section on using device simulation to debug your scripts. Then try running this script while watching the TSX camera window to verify your work.

Using Device Simulators for Debugging

Often times I find it inconvenient to test scripts on my actual system. Software Bisque has graciously provided an alternative – device simulators. I set up a debugging environment in TSX as follows:

1) Create a new Profile ("File" -> "New Profile", name it "Simulator.ini" then "Save")
2) For every device,
    a. Select the device Set Up.
    b. Pull down the device Setup menu list and select "Choose"
    c. In the device listing, select "Software Bisque" and then the device's "Simulator"
    d. Close the device Set Up window
3) Once created, the simulator profile can be opened with the "File" command.

The simulated devices may not support all the functions of every device. For instance, the telescope simulator doesn't home or toggle tracking. But, they do enough to get much of any script debugged.

Final Project: Mount Automation VBScript

In this extra credit exercise, consider a script to do the following:

1) Create a connection to the TSX telescope class
2) Connect the TSX to the telescope
3) Move the telescope to its park position and park
4) Create a connection to the TSX star chart class
5) Find the coordinates of the star "Vega" 6) Unpark the telescope
7) Slew the telescope to the star "Vega"
8) Announce the accomplishment

Here's the script that does all this. I'm not going to explain all this at all. Just know that it works. It's your opportunity to use *ScriptTheSkyX*™ to figure it out – that's the exercise. So, clear your "firsttry.vbs" with Notepad and enter these statements, then go look it up:

```
set tsxt = CreateObject("TheSkyX.sky6RASCOMTele")
status = tsxt.Connect()
status = tsxt.Park()

tgtname  = "Vega"
set tsxs = CreateObject("TheSkyX.sky6StarChart")
status = tsxs.Find (tgtname)

set tsxo = CreateObject("TheSkyX.sky6ObjectInformation")
tsxo.Index = 0
status = tsxo.Property (54)
tgtRA = tsxo.ObjInfoPropOut
status = tsxo.Property (55)
tgtDec = tsxo.ObjInfoPropOut

status = tsxt.Connect()
status = tsxt.UnPark()
status = tsxt.SlewToRADec  (tgtRA, tgtDec, tgtname)

msgbox("All Done")
```

Part III: VBS Scripting Examples

The following section describes how a few VBS examples work.

Autofocus Script

set tsx_cc = CreateObject("TheSkyX.ccdsoftCamera")	'Create connector for ccdsoftCamera class
iCamStat = tsx_cc.Connect()	'Connect to camera device
iFocStat = tsx_cc.focConnect()	'Connect to focuser device
tsx_cc.FilterIndexZeroBased = 3	'Set the (zero-based) filter number to fourth filter
tsx_cc.ImageReduction = 3	'AutoDark reduction
tsx_cc.ExposureTime = 10	'Ten second exposure
tsx_cc.Delay = 0	'No delay
iFocStatus = tsx_cc.AtFocus2()	'Run @Focus2

Autoguiding Script

cdStateAutoGuide = 5	'Set constant for autoguide enumeration
cdStateCalibrate = 6	'Set constant for
cdAutoDark = 3	'Set constant for reduction enumeration
set tsx_ag = CreateObject("TheSkyX.ccdsoftCamera")	'Create connection to ccdsoftCamera class
tsx_ag.Autoguider = 1	'Direct this connection to refer to the autoguider camera
iCamStat = tsx_ag.Connect()	'Connect to the autoguider camera device
If tsx_ag.Connect() = 0 Then	'Check to see if guide camera is connected, if so…
If (tsx_ag.state = cdStateAutoGuide Or tsx_ag.state =cdStateCalibrate) Then	'Find out with the guide camera is up to
tsx_ag.abort()	'Abort if currently autoguiding or calibrating
Else	
iCamsat = tsx.ag.Autoguide()	'Start autoguiding if not
End If	
End If	

Automated Image Link Settings Script

set tsx_ails = CreateObject("TheSkyX.AutomatedImageLinkSettings")	'Create connection to Automated Image Link Settings class
dIScale = tsx_ails.imageScale	'This property holds the image scale.
dPA = tsx_ails.positionAngle	'This property holds the position angle.
dExposure = tsx_ails.exposureTimeAILS	'This property holds the exposure time for Closed Loop Slew and T-point runs
iFOVSearch = tsx_ails.fovsToSearch	'This property holds the number of field of views to search while Image Linking
iRetry = tsx_ails.retries	'This property holds the number of ImageLink retries upon failure.
MsgBox("Automated Image Link Settings:" & vbCrLf & vbCrLf & _	'Write out the five parameters
"Scale: " & FormatNumber(dIScale) & vbCrLf & _	' note that the underscore character
"Position Angle: " & FormatNumber(dPA) & vbCrLf & _	' is used for line continuation
"Exposure: " & FormatNumber(dExposure) & vbCrLf & _	
"FOVs to Search: " & FormatNumber(iFOVSearch) & vbCrLf & _	
"Retries: " & FormatNumber(iRetry))	

Automated Search Script

sYourUserName = "Rick"	'Change to current username to create correct save file location
szPathToMapFile = "C:\Users\" & sYourUserName & _ "\Documents\Software Bisque\TheSkyX Professional Edition\Exported Data\map.txt"	'Use TheSky to generate a text file of mapping points
dExposure = 1.0	'Set the exposure time for the image
set tsx_tele = CreateObject("TheSkyX.Sky6RASCOMTele")	'Create Objects
set tsx_cam = CreateObject("TheSkyX.ccdSoftCamera")	
set tsx_util = CreateObject("TheSkyX.Sky6Utils")	
tsx_tele.Connect()	'Connect mount
tsx_cam.Connect()	‘Connect camera
set MyFileObject = CreateObject("Scripting.FileSystemObject")	‘Create a class object to read a file
set MyFile = MyFileObject.GetFile(szPathToMapFile)	'Open the observing list export file for targets
set MyFileStream = MyFile.OpenAsTextStream	'Stream object for Export Data text file
LineFromFile = MyFileStream.ReadLine	'Get the first line -- headers
If LineFromFile = "" Then	'Exit if the file is empty
Return	
End If	
Do	
iRAindex = InStr(LineFromFile, "RA")	
iDecindex = InStr(LineFromFile, "Dec")	
MsgBox("RA: " + Mid(LineFromFile, iRAindex, 13) + _ " Dec: " + Mid(LineFromFile, iDecindex, 13))	
sname = Left(LineFromFile, 12)	
tsx_util.ConvertStringToRA(Mid(LineFromFile, iRAindex, 13))	
dAz = tsx_util.dOut0	
tsx_util.ConvertStringToDec(Mid(LineFromFile, iDecindex, 13))	
dAlt = tsx_util.dOut0	
tsx_tele.SlewToAzAlt dAz, dAlt, sname	'Slew to object
tsx_cam.ExposureTime = dExposure	'Set exposure time
tsx_cam.AutoSaveOn = True	
tsx_cam.TakeImage()	‘and try for image
LineFromFile = MyFileStream.ReadLine	
If LineFromFile = "" Then	'Exit if the file is empty
Return	
End If	
Loop	‘End of loop

Camera Script

cdLight = 1	‘Constant for frame type enumeration
cdAutoDark = 3	‘Constant for image reduction enumeration
set tsx_cam = CreateObject("TheSkyX.ccdSoftCamera")	'Create object for ccdSoftCamera class
set tsx_img = CreateObject("TheSkyX.ccdSoftImage")	‘Create object for ccdSoftImage class
tsx_cam.Connect()	'Connect TSX to the camera
tsx_cam.ExposureTime = 1.0	'Set the exposure time
tsx_cam.Delay = 0.0	'Set an exposure delay
tsx_cam.Frame = cdLight	‘Set for Light frame
tsx_cam.ImageReduction = cdAutoDark	'Set for Autodark reduction
tsx_cam.Asynchronous = False	'Set for synchronous imaging (wait until done)
iCamStatus = tsx_cam.TakeImage()	'Take image
If iCamStatus <> 0 Then	‘Check for error
Msgbox ("Cam Error: " & FormatNumber(iCamStatus))	‘If so, report it

End If	

Closed Loop Slew Script

cdLight = 1	‘Constant for frame type emumeration
cdAutoDark = 3	‘Constant for image reduction enumeration
set tsx_cam = CreateObject("TheSkyX.ccdSoftCamera")	'Create object for ccdSoftCamera class
set tsx_cls = CreateObject("TheSkyX.ClosedLoopSlew")	‘Create object for ClosedLoopSlew class
tsx_cam.Connect()	'Connect TSX to the camera
tsx_cam.ExposureTime = 10.0	'Set the exposure time
tsx_cam.Delay = 5.0	'Set an exposure delay
tsx_cam.Frame = cdLight	'Set a frame type
tsx_cam.ImageReduction = cdAutoDark	'Set for autodark
tsx_cam.Asynchronous = False	'Set for synchronous imaging (wait until done)
tsx_sc.Find("M39")	‘Find M39
clsstat = tsx_cls.exec()	'Execute CLS

Data Wizard Script

sk6ObjInfoProp_NAME1 = 0	‘Constant for Information Property enumeration
sk6ObjInfoProp_RA_2000 = 56	‘Constant for Information Property enumeration
sk6ObjInfoProp_DEC_2000 = 57	‘Constant for Information Property enumeration
sUserName = "Rick	'Username for user directory: insert your own
set tsx_dw = CreateObject("TheSkyX.Sky6DataWizard")	'Create object Datawizard object
'set tsx_oi = CreateObject("TheSkyX.Sky6ObjectInformation")	'Create Object Information object
tsx_dw.Path = "C:\Users\" & sUserName & "\Documents\Software Bisque\TheSkyX Professional Edition\Database Queries\Bright objects visible now.dbq"	'Set query path to a user-defined test query database for this example. If you don't have it, make it. Best if the Messier catalog is selected.
tsx_dw.Open()	‘Open query database file
set tsx_oi = tsx_dw.RunQuery	‘Run query: tsx_oi will bei an array of object information indexed by the tsx_oi.Index property
For i = 0 to To tsx_oi.Count - 1	'For each object information in the list, get the name, RA and Dec..
tsx_oi.Index = i	
tsx_oi.Property(sk6ObjInfoProp_NAME1)	
sname = tsx_oi.ObjInfoPropOut	
tsx_oi.Property(sk6ObjInfoProp_RA_2000)	
sRA = tsx_oi.ObjInfoPropOut	
tsx_oi.Property(sk6ObjInfoProp_DEC_2000)	
sDec = tsx_oi.ObjInfoPropOut	
MsgBox(sname & " " & sRA & " " & sDec)	‘Print name, RA and Dec
Next	

Telescope (Mount) Script

set tsx_ts = CreateObject("TheSkyX.Sky6RASCOMTele")	'Create the telescope class object
tsx_ts.Connect()	'Connect to the telescope
tsx_ts.GetRaDec()	‘Acquire telescope RA/Dec coordinates (two step process)
tRA = tsx_ts.RA()	‘Get RA
tDec = tsx_ts.Dec()	‘Get Dec
tsx_ts.GetAzAlt()	‘Acquire telescope Alt/Az coordinates (two step process)
tAlt = tsx_ts.Alt()	‘Get Altitude
tAzm = tsx_ts.Az()	‘Get Azimuth
tsx_ts.SlewToRaDec(2.0, 43.0, "Home")	'Slew to an arbitrary RA and Dec
tsx_ts.Sync (3.0, 43.0, "Matt")	‘Synchronize on an arbitrary point
tsx_ts.Disconnect()	‘Disconnect

Dome Control Script

tsx_do = CreateObject("TheSkyX.Sky6Dome")	‘Create Dome class object
tsx_do.Connect()	‘Connect TSX to the dome driver
tx_do.OpenSlit()	‘Open the dome slit
tsx_do.GetAzAl()	‘Initiate a read the current Azimuth and Elevation
DomeAz = tsx_do.dAz	‘Get the azimuth
DomeEl = tsx_do.dEl	‘Get the elevation
tsx_do.GoToAzEl 0, 45	'Rotate the dome at az = 0, alt = 45
tsx_do.FindHome()	‘Rotate the dome to the home position
tsx_do.CloseSlit()	‘Close the dome slit
tsx_do.Disconnect()	'Disconnect the dome

Filter Control Script

cdLight = 1	‘Constant for image type enumeration
cdAutoDark = 3	‘Constant for image reduction enumeration
dExposure = 3.0	‘Constant for exposure length (seconds)
ifilter = 3	'Constant for 4nd filter slot, probably clear/lumenscent
set tsx_cc = CreateObject("theskyx.ccdsoftcamera")	'Create camera class object
tsx_cc.Connect()	‘Connect TSX to camera
tsx_cc.ExposureTime = dExposure	'Set exposure length
tsx_cc.Frame = cdLight	'Set frame type to Light frame
tsx_cc.FilterIndexZeroBased = ifilter	'Set filter
tsx_cc.Delay = 5.0	'Set preexposure delay for filter change at five seconds
tsx_cc.Asynchronous = False	'Set method type to Synchronous
tsx_cc.ImageReduction = cdAutoDark	'Set for autodark
tsx_cc.TakeImage()	'Take image
tsx_cc.Disconnect()	‘Disconnect TSX from camera

Image Link Script

sUserName = "Rick"	'Username for documents directory -- change accordingly
sPathName = "C:\Users\" & sUserName & "\Documents\Software Bisque\TheSkyX Professional Edition\Camera AutoSave\Temp\temp.fit"	'FITS pathname
set tsx_il = CreateObject("TheSkyX.ImageLink")	'Create image link class object
set tsx_ilr = CreateObject("TheSkyX.ImageLinkResults")	'Create image link results class object
tsx_il.PathToFITS = sPathName	'Set path to file
tsx_il.execute()	'Run ImageLink
If tsx_ilr.Succeeded = 0 Then	'Check on result
MsgBox("Error: " + FormatNumber(tsx_ilr.ErrorCode) & " " & tsx_ilr.errortext)	'Did not succeed: print out error information
Else	
MsgBox("RA: " + FormatNumber(tsx_ilr.imageCenterRAJ2000) + " Dec: " + FormatNumber(tsx_ilr.imageCenterDecJ2000))	'Print the image center location
End If	

Field of View Script

sk6MyFOVProp_PositionAngleDegrees = 1	'Constant for MyFOV enumeration -- PA
iFOV = 0 'First FOV definition	'Constant for first FOV definition
set tsx_fov = CreateObject("TheSkyX.Sky6MyFOVs")	'Create FOV class object
set tsx_sc = CreateObject("TheSkyX.Sky6StarChart")	'Create StarChart class object
tsx_fov.Name(iFOV)	'Get FOV name (two step process)
fovname = tsx_fov.OutString	
tsx_fov.Property (fovname, _ 0, _ sk6MyFOVProp_PositionAngleDegrees)	'Get FOV PA (two step process)
fovPA = tsx_fov.OutVar	

List Search Script

sk6ObjInfoProp_ALT = 58	‘Constant for Information Property Altitude enumeration
sk6ObjInfoProp_AZM = 59	‘Constant for Information Property Azimuth enumeration
dExposure = 1.0	'Set the exposure time for the image
dim targetlist(6)	'Create a Target List array of names
targetlist(1) = "NGC1348"	
targetlist(2) = "NGC1491"	
targetlist(3) = "NGC179"	
targetlist(4) = "NGC1798"	
targetlist(5) = "M39"	
targetlist(6) = "NGC2165"	
set objChrt = CreateObject("TheSkyX.Sky6StarChart")	'Create StarChart class object
set objTele = CreateObject("TheSkyX.Sky6RASCOMTele")	‘Create Telescope class object
set objCam = CreateObject("TheSkyX.ccdSoftCamera")	‘Create Camera class object
set objUtil = CreateObject("TheSkyX.Sky6Utils")	‘Create Utility class object
set objInfo = CreateObject("TheSkyX.Sky6ObjectInformation")	‘Create Object Information class object
objTele.Connect()	‘Connect TSX to mount
objCam.Connect()	‘Connect TSX to camera
For i = 1 to 6	‘For each of the targets in the target list array
tname = targetlist(i)	‘ Get the target name
objChrt.Find(tname)	‘ Find the target
objInfo.Property(sk6ObjInfoProp_ALT)	‘ Set object info for “altitude”
dAlt = objInfo.ObjInfoPropOut	‘ Get the altitude
objInfo.Property(sk6ObjInfoProp_AZM)	‘ Set object info for ‘azimuth”
dAz = objInfo.ObjInfoPropOut	‘ Get the azimuth
objTele.SlewToAzAlt dAz, dAlt, tname	‘ Slew mount to
objCam.ExposureTime = dExposure	' Set exposure time
objCam.TakeImage()	‘ Take Image
Next	
objTele.Disconnect()	‘Disconnect mount
objCam.Disconnect()	‘Disconnect camera

Star Chart I Script

sk6DocProp_UseComputerClock = 5	‘Constant for Document Property enumeration
sk6DocProp_JulianDateNow = 9	‘Constant for Document Property enumeration
sk6DocProp_Time_Zone = 2	‘Constant for Document Property enumeration
sk6DocProp_ElevationInMeters = 3	‘Constant for Document Property enumeration
sk6DocProp_Latitude = 0	‘Constant for Document Property enumeration
sk6DocProp_Longitude = 1	‘Constant for Document Property enumeration
sk6DocProp_LocationDescription = 63	‘Constant for Document Property enumeration
dJD = 2452066.0	'=06/05/2001, ignored if UseCompterClock=1
dTZ = 7	'MST
dElev = 1000	'meters
bUseComputerClock = 1	'1=Yes 0=No
dLat = 39.5	‘Latitude = 39.5 (north)
dLong = 105.5	‘Longitude – 105.5 (west)
szLoc = "Location from script"	‘Location description string
set tsx_ts = CreateObject("TheSkyX.Sky6RASCOMTele")	‘Create telescope class object
set tsx_sc = CreateObject("TheSkyX.Sky6StarChart")	‘Create star chart class object
tsx_ts.Disconnect()	‘Disconnect telescope (to enable time changes)
tsx_sc.SetDocumentProperty (sk6DocProp_UseComputerClock, 0)	‘Turn off the computer clock
tsx_sc.SetDocumentProperty sk6DocProp_JulianDateNow, dJD	'Change the julian date
tsx_sc.SetDocumentProperty sk6DocProp_Time_Zone, dTZ	'Change the time zone
tsx_sc.SetDocumentProperty sk6DocProp_ElevationInMeters, dElev	'Change the elevation
tsx_sc.SetDocumentProperty sk6DocProp_Latitude, dLat	'Change the latitude
tsx_sc.SetDocumentProperty sk6DocProp_Longitude, dLong	'Change the longitude
tsx_sc.SetDocumentProperty sk6DocProp_LocationDescription, szLoc	'Change the location description

Star Chart II Script

set tsx_sc = CreateObject("TheSkyX.Sky6StarChart")	'Create Star Chart class object
set tsx_oi = CreateObject("TheSkyX.Sky6ObjectInformation")	‘Create Object Information class object
sname = "M51"	‘Set target name string
tsx_sc.Find(sname)	‘Look up the target
tsx_oi.Property(sk6ObjInfoProp_RA_2000)	‘Get RA (two step process)
dRA = tsx_oi.ObjInfoPropOut	
tsx_oi.Property(sk6ObjInfoProp_DEC_2000)	‘Get Dec (two step process)
dDec = tsx_oi.ObjInfoPropOut	
tsx_oi.Property(sk6ObjInfoProp_AZM)	‘Get azimuth (two step process)
dAz = tsx_oi.ObjInfoPropOut	
tsx_oi.Property(sk6ObjInfoProp_ALT)	‘Get altitude (two step process
dAlt = tsx_oi.ObjInfoPropOut	
MsgBox("ScreenRa=" & FormatNumber(tsx_sc.RightAscension))	‘Get and display RA of chart center
MsgBox("ScreenDec=" & FormatNumber(tsx_sc.Declination))	‘Get and display Dec of chart center
MsgBox("ScreenFOV =" & FormatNumber(tsx_sc.FieldOfView))	‘Get and display FOV of chart
MsgBox("ScreenRotation =" & FormatNumber(tsx_sc.Rotation))	‘Get and display Rotation of chart
tsx_sc.FieldOfView = 10.0	‘Set FOV to 10 degrees
tsx_sc.Rotation = 300.0	'Set Star Chart Rotation to 300 degrees