**VBScript Starter Kit: Simple automation scripting for *TheSky64****™*

*Rick McAlister, Rev 2.1, February 2019*

**Overview**

Do you want to automate *TheSky64™* 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 *TheSky64™* 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: Scripting Environment**

A diagram of a software application

Description automatically generated

Scripts are generally just short series of instructions which are run to execute a simple sequence of functions within some much more complicated application. Often times a script merely automates a set of steps that a user might do manually. Scripts can be run stand-alone or launched from some other application. For instance, an automation application like SGP might launch a script at upon completion of an imaging sequence to park the mount and cool down the camera.

As you shall see, a VBS script is a text file with the extension “.vbs”. Windows has a built-in capability called *Windows Script Host* that translates text in a .vbs file into commands and directs them through an inter-application interface (COM) to be executed by other applications. The text instructions tell the *Windows Script Host* which application to send the commands to and what to say. That’s it.

One note, though. For TheSky to be accessed by COM interface, it must be run once (but only once) in Administrator

Mode so that it can tell the COM interface how to get a hold of it, kind of like trading phone numbers. Now, on to the good stuff.

**Part II: The Tools**

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

1. *ScriptTheSky64™* 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. *ScriptTheSky64™:* 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, *ScriptTheSky64™* 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/scriptTheSky64.](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 *ScriptTheSky64™*, 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). *ScriptTheSky64™* 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. *ScriptTheSky64™* 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 *ScriptTheSky64™* 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 *ScriptTheSky64™*.

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 III: 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("TheSky64.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 “TheSky64” 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 *ScriptTheSky64™* 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(“TheSky64.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,
   1. Select the device Set Up.
   2. Pull down the device Setup menu list and select “Choose”
   3. In the device listing, select “Software Bisque” and then the device’s “Simulator”
   4. 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
6. Slew the telescope to the star “Vega”
7. 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 *ScriptTheSky64™* 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("TheSky64.sky6RASCOMTele")

status = tsxt.Connect()

status = tsxt.Park()

tgtname = "Vega"

set tsxs = CreateObject("TheSky64.sky6StarChart")

status = tsxs.Find (tgtname)

set tsxo = CreateObject("TheSky64.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("TheSky64.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("TheSky64.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("TheSky64.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\TheSky64 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("TheSky64.Sky6RASCOMTele") | 'Create Objects |
| set tsx\_cam = CreateObject("TheSky64.ccdSoftCamera") |  |
| set tsx\_util = CreateObject("TheSky64.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("TheSky64.ccdSoftCamera") | 'Create object for ccdSoftCamera class |
| set tsx\_img = CreateObject("TheSky64.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("TheSky64.ccdSoftCamera") | 'Create object for ccdSoftCamera class |
| set tsx\_cls = CreateObject("TheSky64.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("TheSky64.Sky6DataWizard") | 'Create object Datawizard object |
| 'set tsx\_oi = CreateObject("TheSky64.Sky6ObjectInformation") | 'Create Object Information object |
| tsx\_dw.Path = "C:\Users\" & sUserName & "\Documents\Software Bisque\TheSky64 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("TheSky64.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("TheSky64.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("TheSky64.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\TheSky64 Professional Edition\Camera AutoSave\Temp\temp.fit" | 'FITS pathname |
| set tsx\_il = CreateObject("TheSky64.ImageLink") | ‘Create image link class object |
| set tsx\_ilr = CreateObject("TheSky64.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("TheSky64.Sky6MyFOVs") | 'Create FOV class object |
| set tsx\_sc = CreateObject("TheSky64.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("TheSky64.Sky6StarChart") | 'Create StarChart class object |
| set objTele = CreateObject("TheSky64.Sky6RASCOMTele") | ‘Create Telescope class object |
| set objCam = CreateObject("TheSky64.ccdSoftCamera") | ‘Create Camera class object |
| set objUtil = CreateObject("TheSky64.Sky6Utils") | ‘Create Utility class object |
| set objInfo = CreateObject("TheSky64.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("TheSky64.Sky6RASCOMTele") | ‘Create telescope class object |
| set tsx\_sc = CreateObject("TheSky64.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("TheSky64.Sky6StarChart") | 'Create Star Chart class object |
| set tsx\_oi = CreateObject("TheSky64.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 |