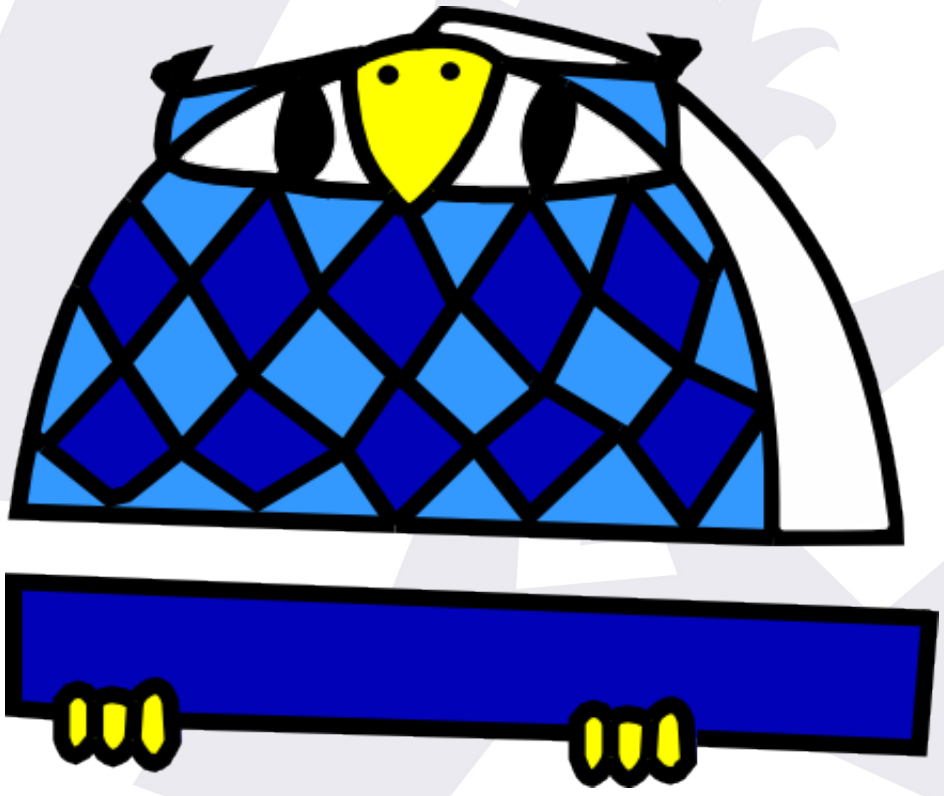
GNS
User's manual



By Lunático Astronomía

INDEX

1. Preface

Thank you for your interest in the **Good Night System**, our innovative product to make your long imaging sessions easier.

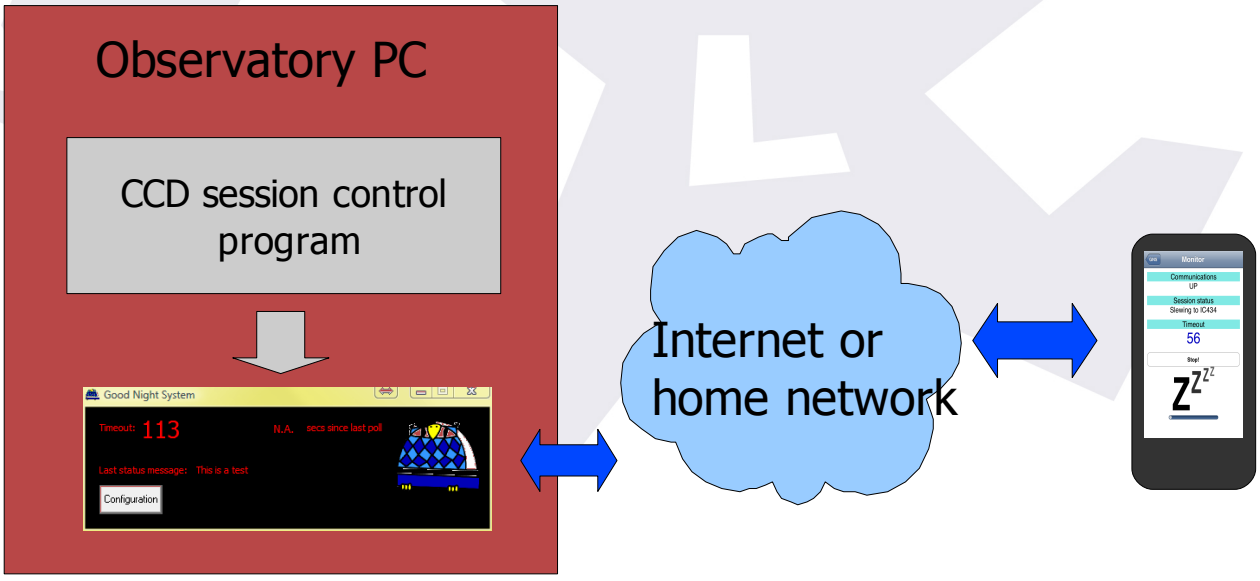
Features that set apart the **Good Night System**:

- **it does not rely on SMSs or emails**; to the contrary, the smartphone constantly plays an active role.
- **safety goes first**: in some cases you may be warned when no real danger exists, most probably if the time allotted for a task is exceeded (because it was too short) or the communications go down; but provided the smartphone is powered, it will wake you up if something goes wrong or *seems to go wrong.*
- both the Windows application and the smartphone app are **small and reliable** by design, and have been **extensively tested**, for a period of several months by a group of real users, under real circumstances.

As you already know, the "**Good Night System**" comprises two elements, both needed to ensure observatory protection:

a) a small windows program
b) a smartphone app

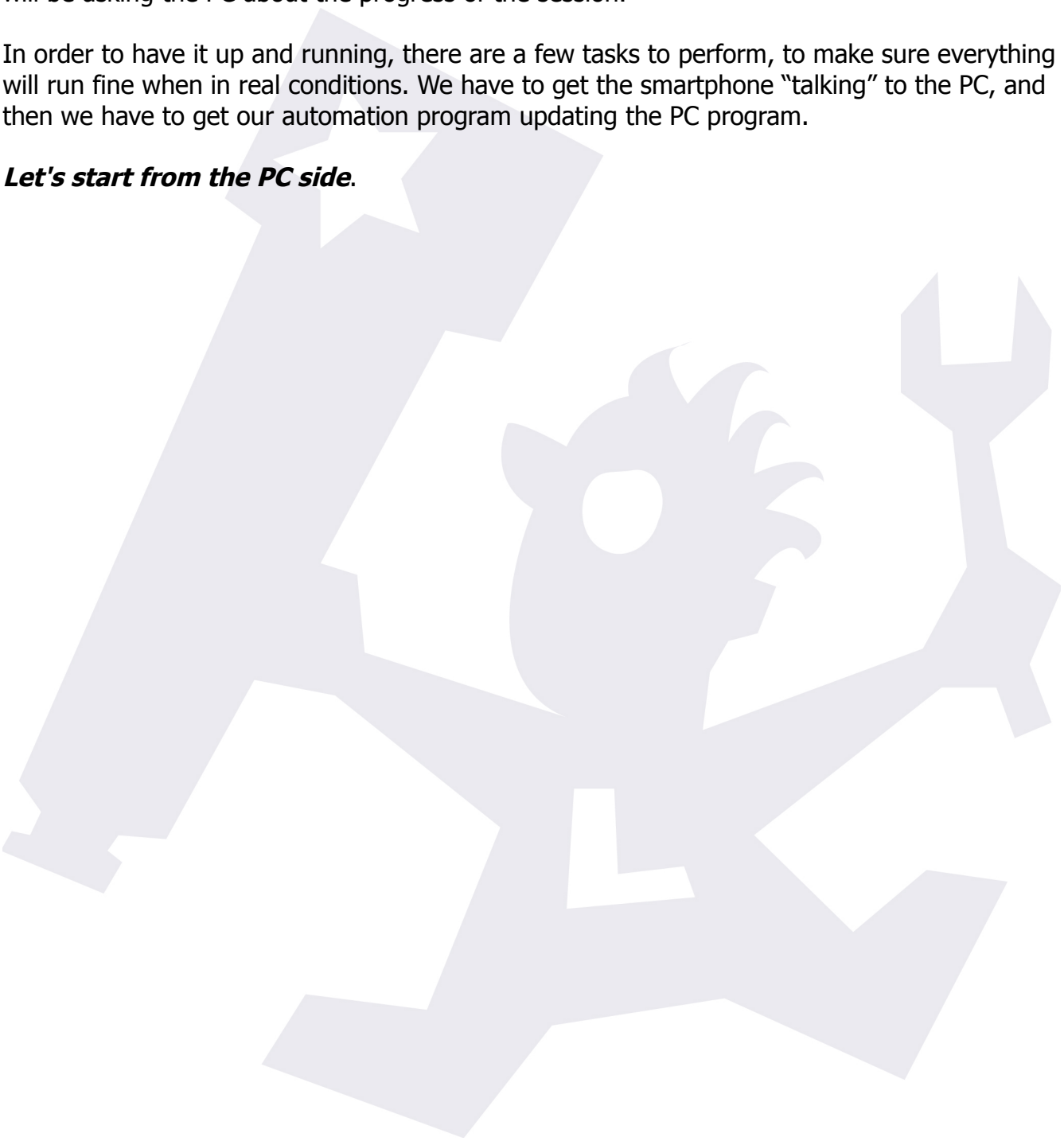
It will work as shown in the following drawing:



So at one side, we'll have our PC GNS program being updated by our ccd/session control software of choice. At the other side, our smartphone (any number of smartphones, actually) will be asking the PC about the progress of the session.

In order to have it up and running, there are a few tasks to perform, to make sure everything will run fine when in real conditions. We have to get the smartphone "talking" to the PC, and then we have to get our automation program updating the PC program.

Let's start from the PC side.



2. Installing and checking the PC program

Brief foreword about 32 and 64 bits systems: the GNS software will run in both systems, always in 32 bits mode – this is not a problem as most, or all, astronomy software that will work with it is 32 bits. In the following procedures you'll find instructions about how to test the software in both kinds of systems.

- **Install the software[1]:** it can be downloaded from the **GNS** page of our web site (http://lunatico.es/site) – the software should install automatically under any "current" Windows version (from XP to Windows 8). The only requisite is to have the .NET platform already installed, which is most likely the case if you are into observatory automation as it is also required by many programs and by the ASCOM platform.

  In case you don't, please visit Microsoft's .NET download page:
  http://www.microsoft.com/net/download
  … and select the platform suitable for your system. **GNS** needs platform v. 2.0.

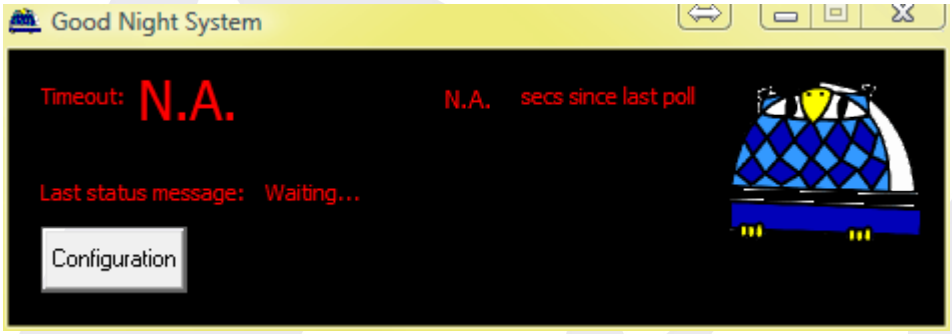
Apart from the main software itself, in the installation folder (c:\program files\GNS, or c:\program files (x86)\GNS, the latter if yours is a 64 bit system), you'll find a few scripts (small programs):

- update.vbs
- alarm.vbs
- switchoff.vbs
- idle.vbs
- message.vbs

These scripts will be used to integrate the **GNS** with some automation systems in the market – other systems have or may have native support.

1 If you plan to use the system with Maxpilote, you'll have to select the Maxpilote folder for installation.

- Now run the software (there's a shortcut from your start menu), and you'll see the main window:

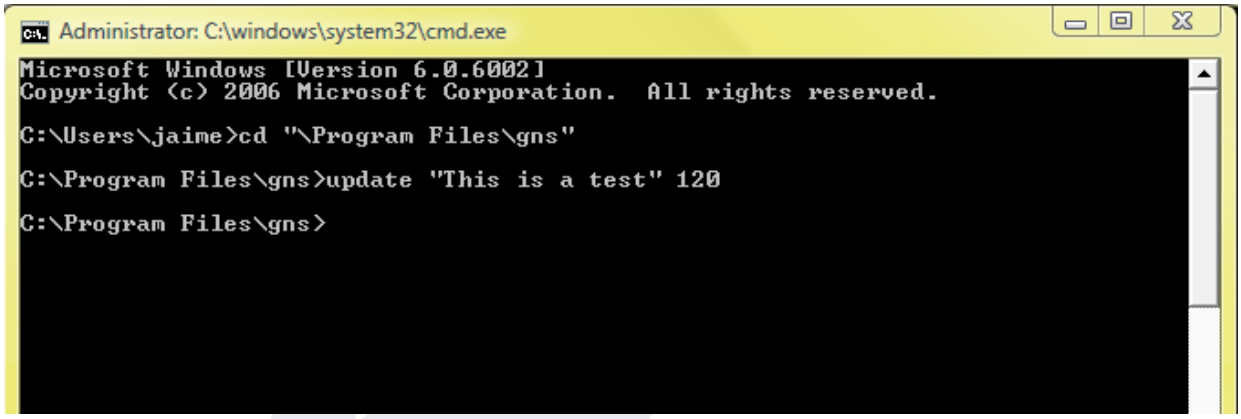


FIREWALL

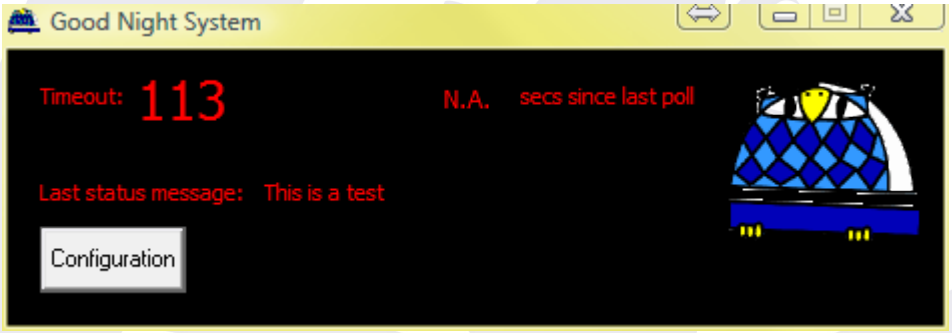
VERY IMPORTANT AT THIS POINT: if you have any kind of firewall installed, it will warn you of the "Good Night System" program (GNSUI.exe) trying to communicate with the outer world. This is normal, since the GNS must start listening for incoming requests, and you should enable (unblock) it in order for the system to work.

- Let's check if the installation went well:
  - Run a command prompt:
    - *For 32 Bits systems*, just: Start menu → All programs → Accessories → Command prompt
    - *For 64 Bits systems*, use the 32 bit version of the command prompt: browse your hard disk to the "Windows\sysWOW64" folder, then launch the "cmd" program found there.
  - Go to the installation folder, type: **cd "\program files\gns"** including the quotes and followed by the return key (again, in 64 bit systems, change to "\program files (x86)\gns").
  - Now type: **update "This is a test" 120** followed by return

– Your command window should look like this:



– And check the **GNS** window did actually update, to show (7 seconds later...):



Note the status message has been updated, … and verify it will count down to 0.

"secs since last poll" remains at **N.A**. as the program has never been contacted by a smartphone; this field will let us know how many seconds have elapsed since the smartphone last requested an update.

Do not close the "cmd" window as we'll use it later.

Please forget the configuration button for the moment.

3. Smartphone installation

Let's go now to the smartphone:

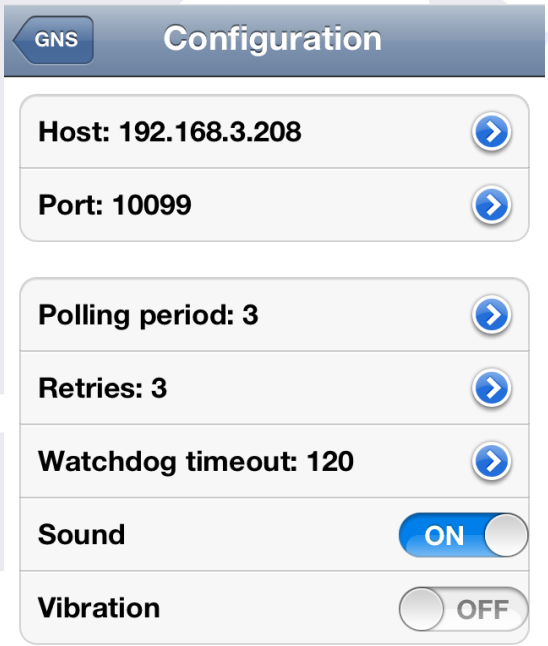
Enable your smartphone network connection (be it wifi or mobile data/3G/4G, whatever).

Download our Free app (from *Google play* for **Android phones**, or from the *AppStore*, in case of **iPhone**).

> **Is is highly recommended to try with the free version first**; it is functionally identical to the full version, with the only exception the watchdog will be disabled and will act as a countdown timer, limiting the monitored session at 30 minutes.
>
> This way you'll be sure everything will run smooth before spending money in the paid version.

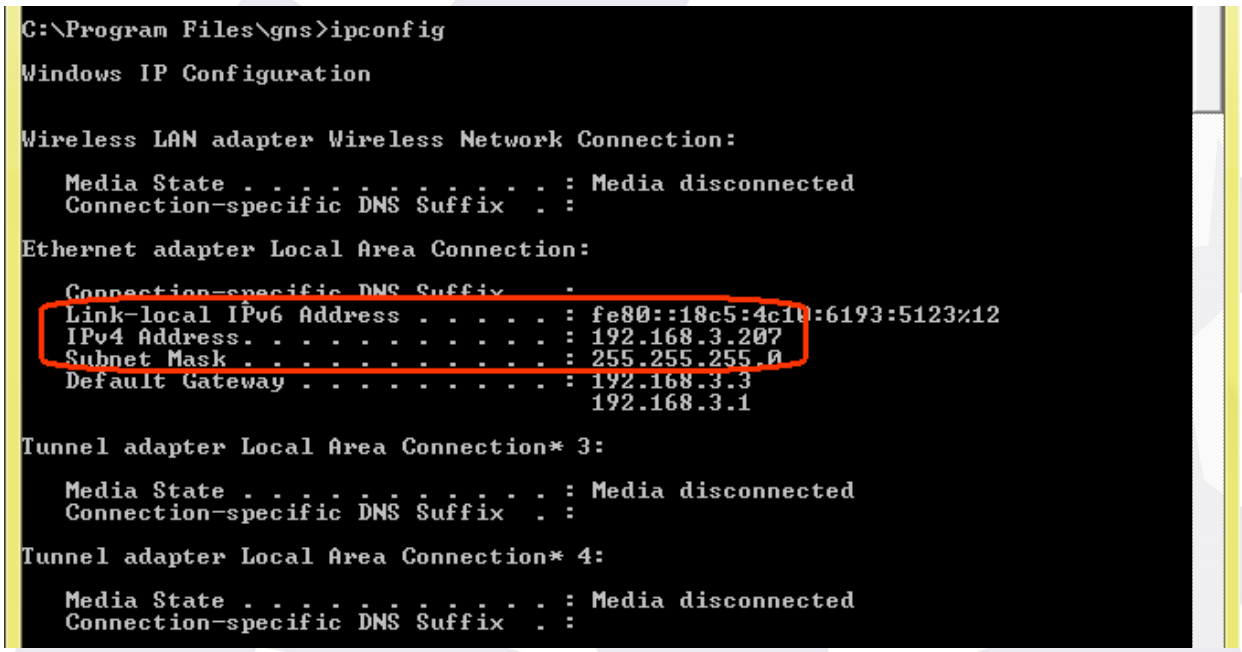
Run the app, and you will get to the initial screen:

Press "Configuration" to open the parameters screen:



At this point, we have to get this app communicating with our PC. Please type the host name (or just the IP address if you are in the same network) of your observatory PC. This can be a bit difficult if you know nothing about networks, but having a remote or automated observatory you most likely know enough.

In case of doubt, go back to your PC, cmd window, and type **ipconfig** followed by return, and look for a meaningful (!) set of 4 dot separated numbers, in a line that will read something like "IPv4 address" (see below):



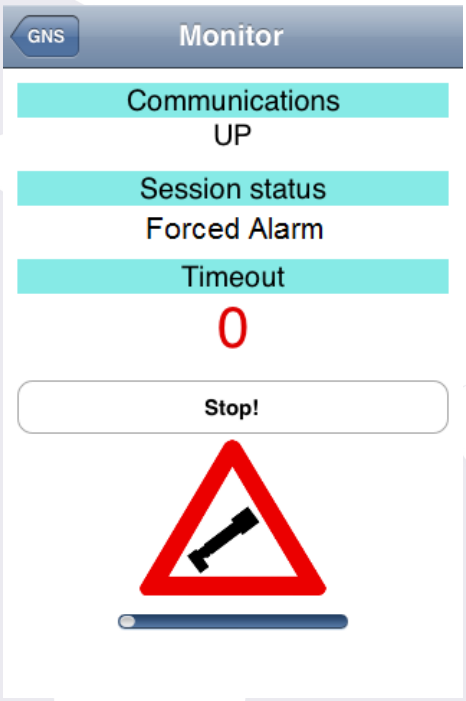
For the time being, leave the rest of fields as they are:

– **Port**: is the tcp/ip "slot" where the requests will be sent (we can change this in the windows application, configuration, in the event the default port is already in use).
– **Polling period**: is the number of seconds the smartphone app will wait after an update to check again.
– **Retries**: means the number of times it will attempt to connect, in case of any problem, before giving up and issuing a warning.
– **Watchdog Timeout**: this is the watchdog (watch-owl?) time. If, for any reason, the smartphone does not get any meaningful communication from the PC for the specified amount of time, it will wake you up.[2]
– **Sound** and **Vibration**, just refer to the means of warning you. Leave at least one active!

Just touch the "**Validate**" button, and from the main screen now press "**Monitor**".

2 Remember the behavior of the watchdog is changed in the free version.

If the network is correctly setup, firewall included, we'll go to the main monitor screen, and in a few seconds the communications will go "UP", and a forced alarm will fire. It is forced because, unless you performed all these steps in less than 2 minutes (we setup a 120 sec time earlier, testing the PC application), the timeout reached 0 long ago.



What if the communications don't go UP and you get other kind of error (watchdog, or "too many conn tries")?

Well, that means the smartphone is not able to talk to the PC. There are a number of possible causes:

– some firewall program is preventing the PC to accept incoming communications; completely disable the firewall and try again. If this works, you should enable the firewall later, but adding a rule, or exception, for the GNSUI.exe program.

– The network settings are not correct. This is a more complex matter. I see three different situations:

a) your have a backyard observatory (or otherwise at home) that shares the same network that the smartphone. You usually control your observatory PC from your living room, using Windows remote desktop or similar software.

This is the most difficult case, as your PC is probably getting it's IP address using DHCP

(a network protocol that assigns a different address every time you power on); that is, in your Windows network settings you've leave the "IP v4 address" setting at the default "automatic".

Using the "ipconfig" program you can find out the current IP address (the 4 number set), but it can change every session, maybe even during an imaging sessions (the address has an expiry time), and this can ruin our purpose. Please do assign your PC a fixed IP address. I'll try to explain how to do this in future updates to this manual, but for the moment please seek assistance if you don't know how to do it.

If this is your case, set the "Host" field of the smartphone configuration to your 4 digit IP address.

b) you have a remote observatory, connecting to it via Internet and using a program such as teamviewer or similar.

In this case the settings are a bit more complex, but you already know or have someone who knows. In a nutshell, you'll need:

– a fixed internal network IP address in your observatory PC (needed for the second step).
– a port "opened" in your observatory router (as to route incoming 10099 TCP port messages to the 10099 port of your PC)
– if your router has a dynamic IP (that is, the public IP address changes from time to time, and every time you power on the router), then you need a dynamic host name, a service provided for free by several network services (such as dyndns.org).

c) your observatory is at home, but in a different network than your mobile.

This is the same, from our point of view, as case (b).

The point of all this is being able to configure your smartphone so its messages will reach the destination PC.

If b or c is your case, please type your complete host name (similar to "myobservatory.dyndns.org") in the "Host" filed of the smartphone configuration.

4. Real life integration

You have surely understood the way it works now; we are going to update the system with every action (or groups of actions, more likely) the scope is going to do, allotting time for them. If the time expires, then an alarm will be issued.

We can also issue an immediate alarm, and of course notify the system of a successful end of the session.

In the simplistic example of the web site – to be honest reflects the approach I'm actually using, and works great – I just update the **GNS**:

- once at the start of the session, with a 10 minute timeout. In this time main imaging should have been reached.
- Once for every imaging group (including telescope slew, plate solve, focus, and actual imaging). The duration of course varies, up to a few hours.
- Last for the end of the session: ccd warm up, telescope park, roof close.

As my observatory is at home, I also launch an immediate alarm in case of unsafe weather detected by my CloudWatcher, but this is a bit of a lazy approach.

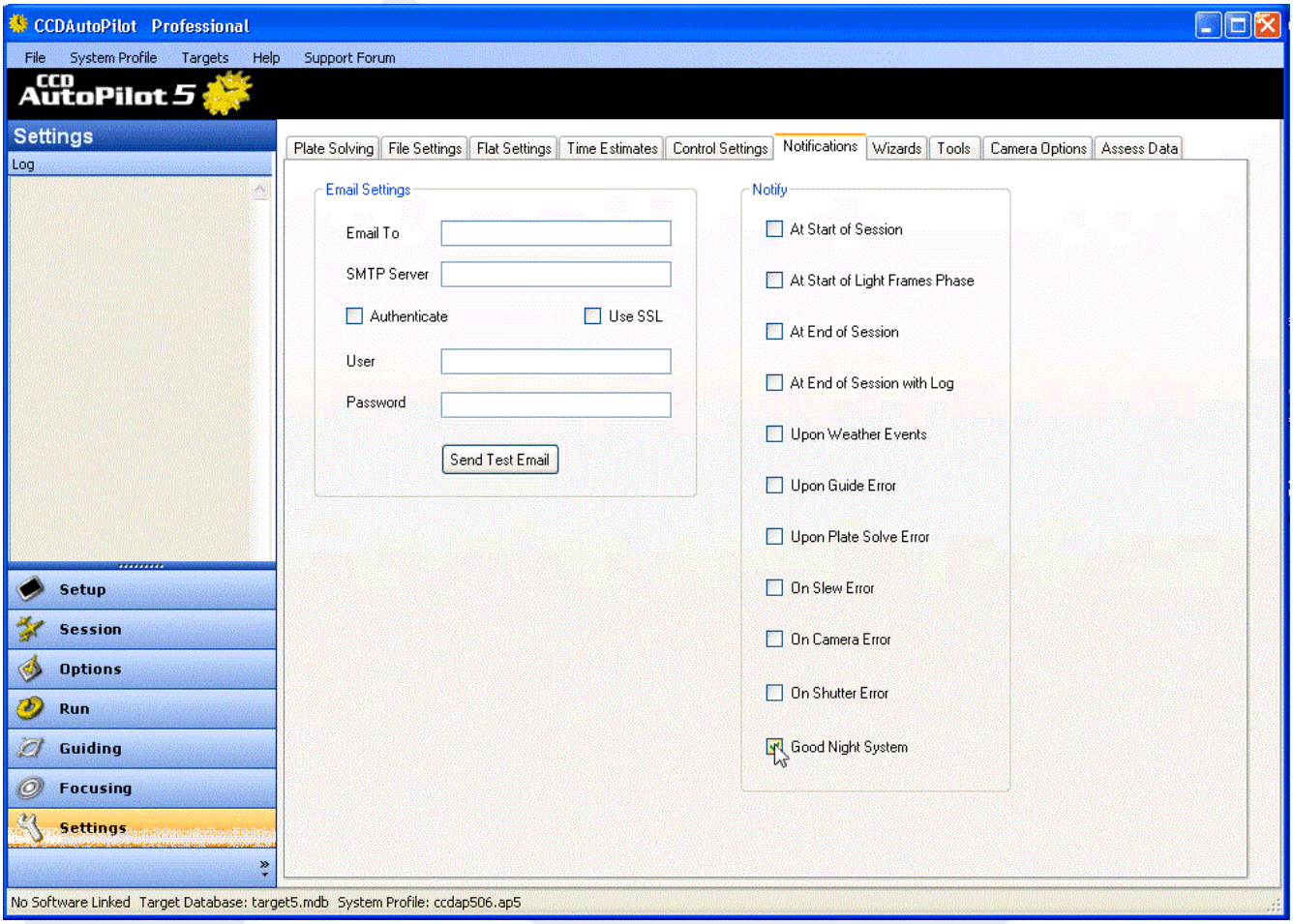
At the moment of this writing, the **GNS** can be used with 4 popular session control systems: *CCDAutoPilot*[3] and *Maxpilote* support the system directly, so no work is needed from your part. Just take a look at next section to see any installation / configuration notes.

CCDCommander and *ACP* can be used quite easily too, but require you to use their scripting features; please see instructions below.

3 CCDAutoPilot support has just been added and tested, but, depending on when you read this, it may have not yet made into the official release.

4.1. CCDAutoPilot 5

As already mentioned, CCDAP v5 includes native support for the GNS (thank you very much John), so you'll just have to activate it from the Settings window, Notifications tab:

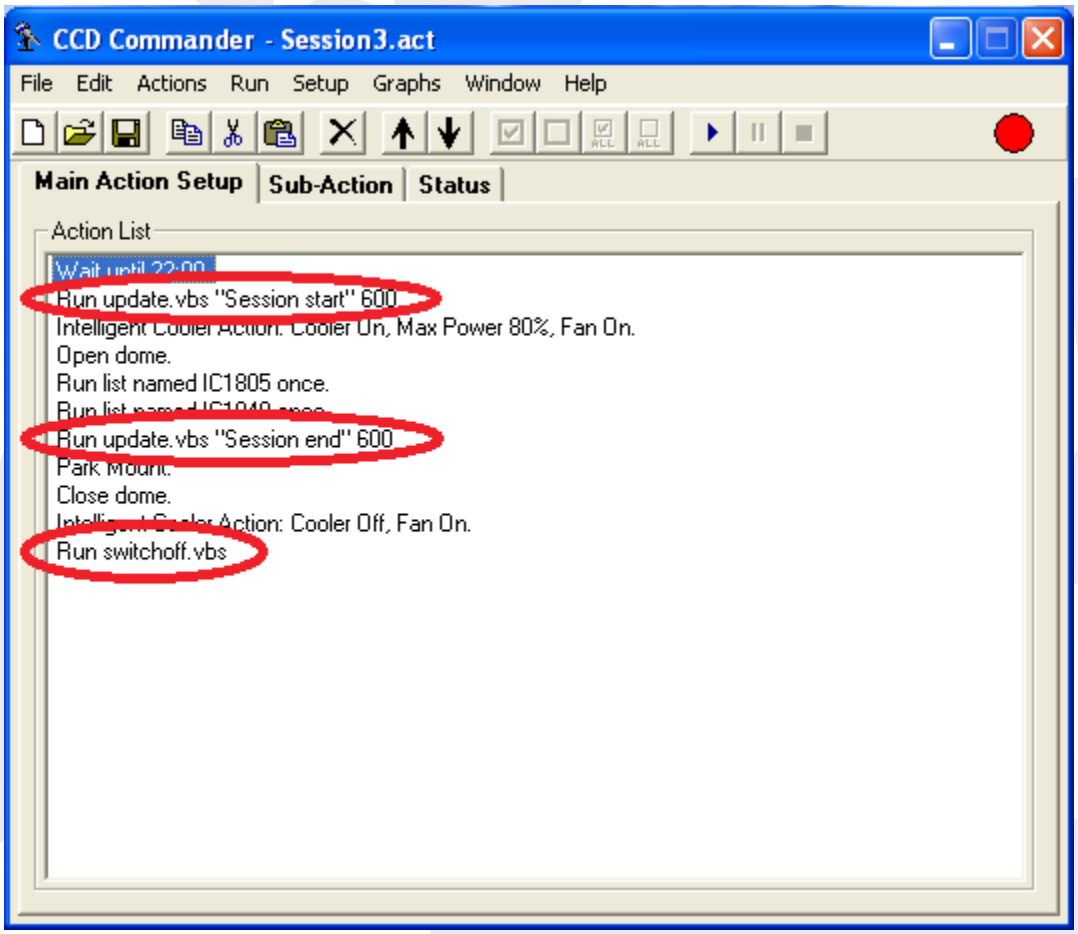


And that's all, the GNS will be launched by CCDAP and your smartphone will track your session!

4.2. CCDCommander

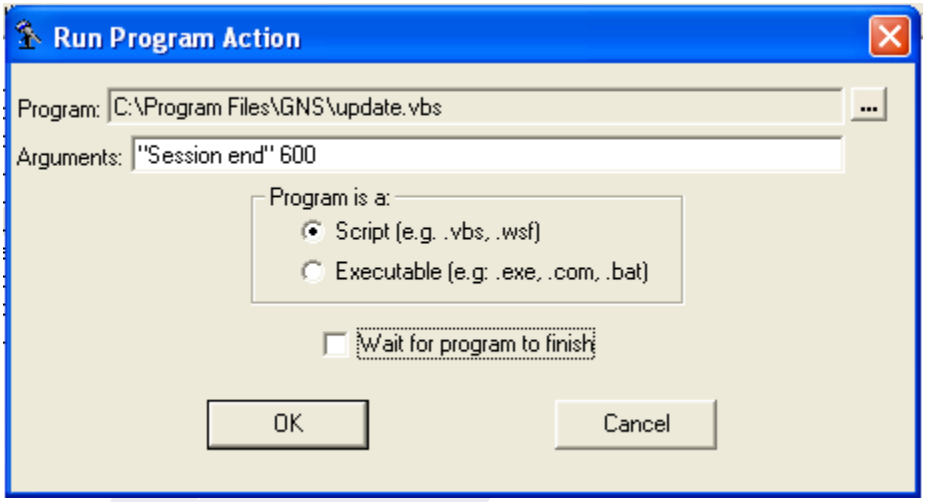
Integrating this system with CCDCommander is really straighforward, as CCDC allows the user to call scripts in many cases and at any point of the action list.

At selected points in your action list, insert a "Call external subprogram" action, to execute the script "update.vbs" included with your **GNS** distribution. You'll have to call it with two parameters, the first one being an informative message, the second one the timeout in seconds. A simple example:



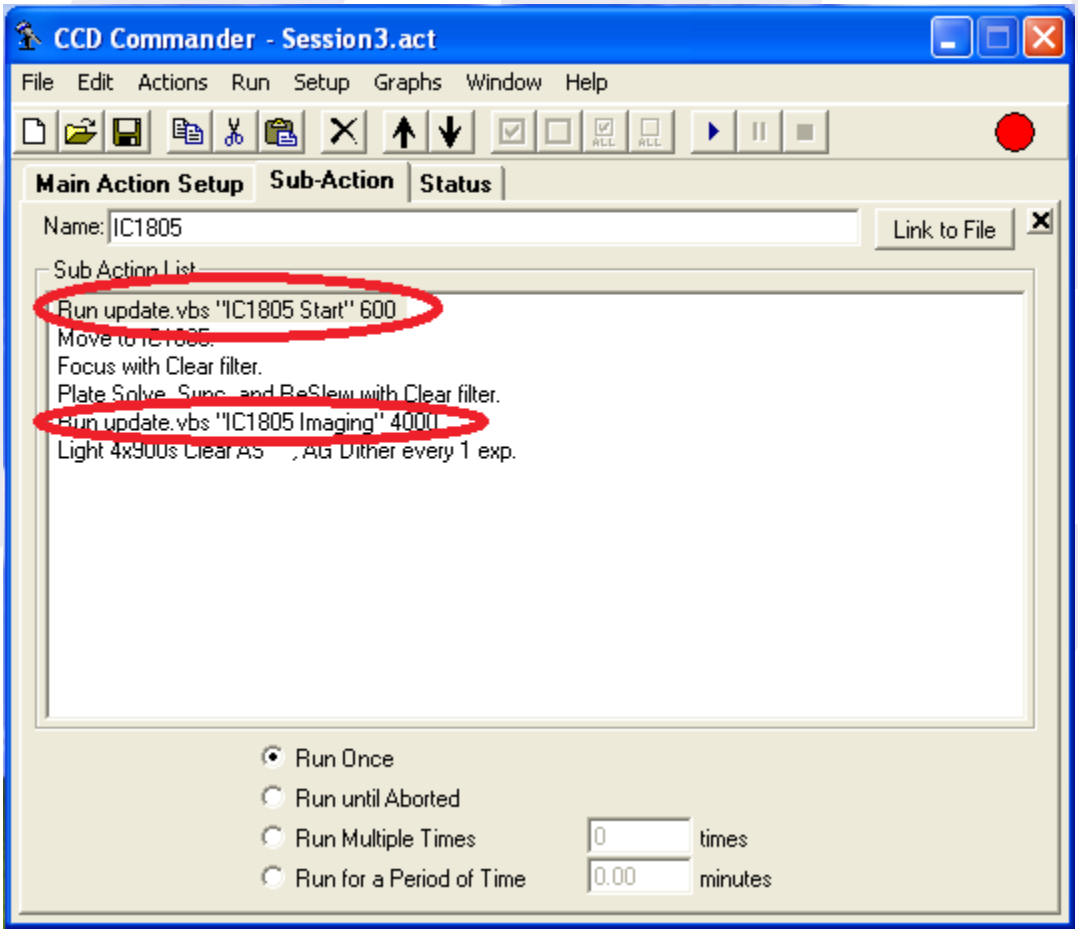
As can be seen, in the "Main Action Setup" I just added two calls to "Update", one at the beginning of the session, another when we start winding up, and a final one to disable the system ("Switchoff").

For each call you'll use the "Run external program" option from the "Actions" menu:



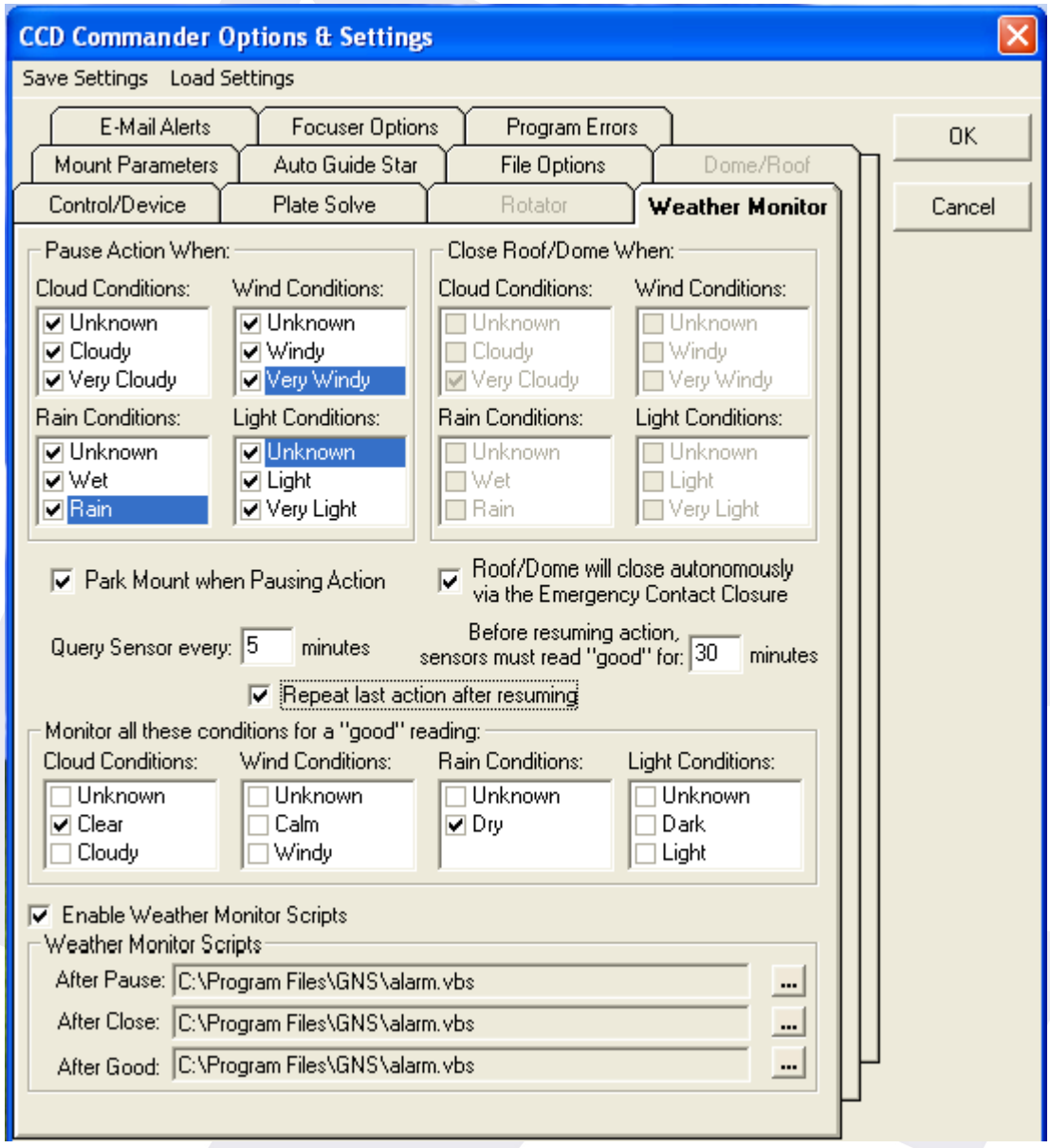
If the message includes more than one word, please enclose it in quotes as in "Session end". Also please note we've selected "Program is a... script", and unchecked (not really important) the optional "Wait for program to finish".

Next, in each Sub-action, you can add as many updates as you wish:



Don't forget to end the main action list calling the script "switchoff.vbs"; this one will inform your smartphone the session ended successfully, and it will stop the communication.

You may also want to call "alarm.vbs" in case of errors, or maybe in case of unsafe weather detected (note: alarm is just an update with a timeout of 0 seconds).



4.3. ACP Observatory Control Software

> *This section on integrating with ACP has been fully written by **Gerald Hitz**, great astronomer, great user, and above all great guy. Thank you!*
>
> *I've only edited it to match the current document style, and changed a reference to the "Idle" script as previously it was called "calm".*
>
> *As a plus, there's also some code for our observatory control product, the Firefly – as Gerald marked it in green, you can safely skip it if you're just interested in the **GNS**.*

4.3.1. ACP Observatory Control Software

The **GNS** can be tightly integrated into ACP with the help of so-called user actions and with extending existing scripts. This short document shows an example of how to change those scripts to interact with Lunatico's **GNS**.

4.3.1. a) ACP User Actions

Please refer to the ACP help how to active user actions. Basically you should find a find a template file for JavaScript and also Visual Basic Script. Take your favorite and create a copy named „UserActions.wsc". After registering the UserActions.wsc file to Windows with regsvr32 (see ACP help, especially if you run a 64bit operating system!), just edit this script.

The red lines have been added to the template, green lines are some examples of how to integrate the Firefly relays and sensor box.

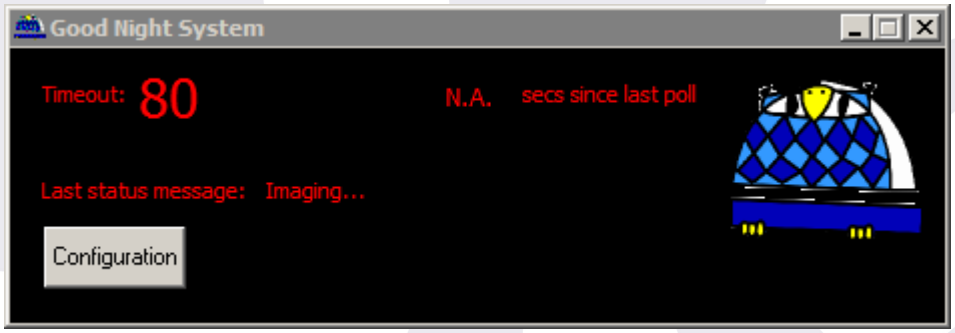
File	*%PROGRAMFILES(X86)%\ACP Obs Control\UserActions.wsc*

```
Function TargetStart(Plan, Target, NextTarget)
  Util.Console.PrintLine "  [useraction] TargetStart called"

        ' Turn Mirror Fan OFF to prevent turbulences while imaging
        Dim FireFly
        set FireFly = CreateObject("FireflyEXP.Help")
        if (FireFly.Connected) then
                while (FireFly.RelayRead( 6 ) = -1)
                        FireFly.RelayOpen( 6 )
                wend
                Util.Console.PrintLine " Mirror fan switched off"
        end if

        Dim wD
```

```
        set wD = CreateObject("GNS.OWL")
        wD.NewMsg = "Target starting"
        wD.NewTimeout = 300
        set wD = nothing

    TargetStart = True
End Function

Function TargetEnd(Plan, Target, NextTarget)
    Util.Console.PrintLine "  [useraction] TargetEnd called"

        Dim wD
        set wD = CreateObject("GNS.OWL")
        wD.NewMsg = "Target finished"
        wD.NewTimeout = -1
        set wD = nothing

    TargetEnd = True
End Function

Function SlewStart(RightAscension, Declination)
        Util.Console.PrintLine "  [useraction] SlewStart called"
        Dim wD
        set wD = CreateObject("GNS.OWL")
        wD.NewMsg = "Slewing..."
        wD.NewTimeout = 180                     'Max. slewing time incl. meridian flip
        set wD = nothing
        SlewStart = True
End Function

Function SlewEnd()
        Util.Console.PrintLine "  [useraction] SlewEnd called"
        Dim wD
        set wD = CreateObject("GNS.OWL")
        wD.NewMsg = "Slew finished"
        wD.NewTimeout = -1                      'Reset slewing timeout
        set wD = nothing
        SlewEnd = True
End Function

Function ImageStart(Interval, Binning, Subframe, FilterNum, ForPointing)
        Util.Console.PrintLine "  [useraction] ImageStart called"
        Dim wD
        set wD = CreateObject("GNS.OWL")
        wD.NewMsg = "Imaging..."
        wD.NewTimeout = Interval + 120          'Add buffer for downloading the image
                                                'Add also some time for the guider settle delay
        set wD = nothing
```

```
        ImageStart = True
End Function

Function ImageEnd(ImageFile)
        Util.Console.PrintLine "  [useraction] ImageEnd called"
        Dim wD
        set wD = CreateObject("GNS.OWL")
        wD.NewMsg = "Imaging finished"
        wD.NewTimeout = -1                              'Reset imaging timeout
        set wD = nothing
        ImageEnd = True
End Function

Function Shutdown()
   Util.Console.PrintLine "  [useraction] Shutdown called"

        Dim wD
        set wD = CreateObject("GNS.OWL")
        wD.NewMsg = "Shutdown"
        wD.NewTimeout = -2
        set wD = nothing

   Shutdown = False                 ' Use built-in shutdown logic
End Function
```

Example screen shot after an exposure has been started.



4.3.1.b) Weather Safety script

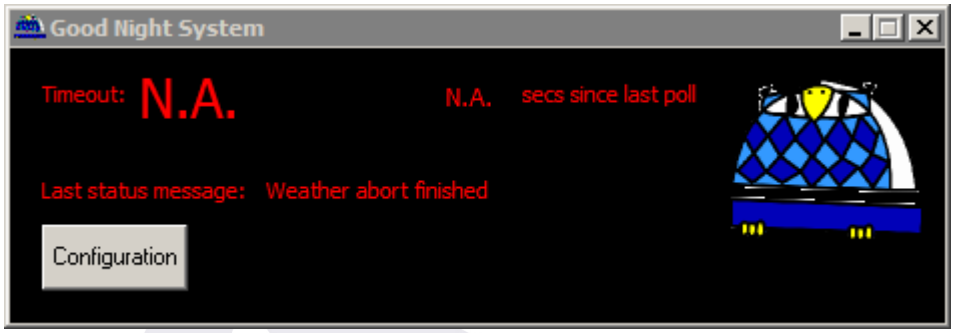
The Weather Safety Script should be changed in a way that the user will be notified if the telescope can't be parked or the dome/roof not closed. If everything is fine, the GNS can be set to „idle" mode at the end of the script.

File	*%PROGRAMFILES(X86)%\ACP Obs Control\ACP-Weather.vbs*

```
Sub Main()
```

```
	Console.PrintLine "Weather Safety Script"
	On Error Resume Next

	Dim wD
	set wD = CreateObject("GNS.OWL")
	wD.NewMsg = "Weather abort..."
	wD.NewTimeout = 300

	if(Util.CameraConnected) then
	  Dim FMX
	  Console.PrintLine("Trying to stop FocusMax")
	  set FMX = CreateObject("FocusMax.FocusControl")
	  FMX.Halt
	  Util.WaitForMilliseconds 1000
	else
	  Console.PrintLine("Camera not connected - nothing to do")
	end if

	Console.PrintLine("Trying to park scope")
	wD.NewMsg = "Parking scope"
	wD.NewTimeout = 300                              'Timeout for parking + roof close

	Telescope.Park

	' Some magic triggers the dome/roof close at this point

	if(Dome.ShutterStatus <> 1) then
		wD.NewMsg = "Alarm! Roof still open"
		wD.NewTimeout = 0
	else
		wD.NewMsg = "Weather abort finished"
		wD.NewTimeout = -1
	end if

	wD.NewMsg = "Weather abort finished"
	wD.NewTimeout = -1
	set wD = nothing

	Console.PrintLine("Finished")
End Sub
```

Screen-shot after weather safety script – Dome is already closed at this time.



4.3.1.c) Autofocus script

A timeout should also be added to the AutoFocus script to catch faults in FocusMax.

File	*%PROGRAMFILES(X86)%\ACP Obs Control\scripts\AutoFocus.vbs*

Somewhere after the filter check (line 49) I added a timeout of 300 seconds

```
Dim wD
set wD = CreateObject("GNS.OWL")
wD.NewMsg = "Autofocus..."
wD.NewTimeout = 300
```

Before terminating this script (look for „SUP.Terminate“) the timer should be cleared

```
wD.NewMsg = "Autofocus finished"
wD.NewTimeout = -1
set wD=nothing
```

4.3.2. ACP Scheduler

One can also edit the Startup and Shutdown scripts from ACP Scheduler to trigger some timeouts here.

4.3.2.a) Startup script

File	*%PROGRAMFILES(X86)%\ACP Scheduler\StartupObs.js*

Find the snippet below and create a timeout of a few minutes in within the intial Autofocus

should start.

```
// ============================
// ADD ANYTHING ELSE YOU NEED
// ============================

Util.ShellExec("C:\\Program Files (x86)\\GNS\\update.vbs", "Startup 300", 1);
```

4.3.2.b) Shutdown script

Here you could initiate the GNS shutdown (instead of UserActions.wsc)

File	*%PROGRAMFILES(X86)%\ACP Scheduler\ShutdownObs.js*

```
// ========================================================
// HERE IS WHERE YOU ADD CODE TO TURN POWER OFF AS NEEDED
// ========================================================

Util.ShellExec("C:\\Program Files (x86)\\GNS\\switchoff.vbs", "", 1);
```

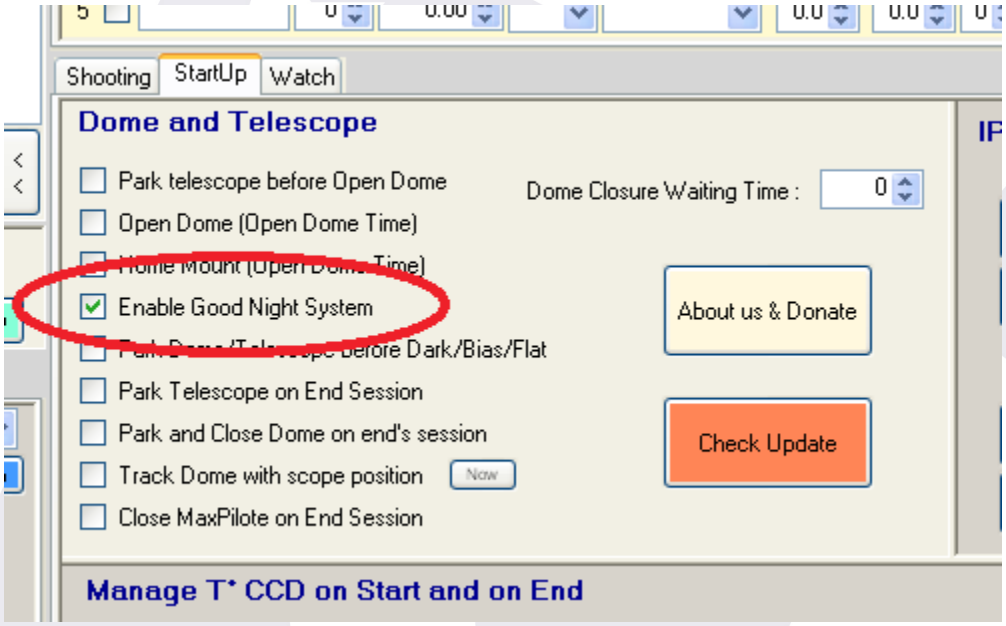
4.3.2.c) Special Autofocus script for Scheduler

If you use ACP Scheduler, the AutoFocus script can be found here:

File	*%PROGRAMFILES(X86)%\ACP Scheduler\AutoFocusScheduler.vbs*

4.4. Maxpilote

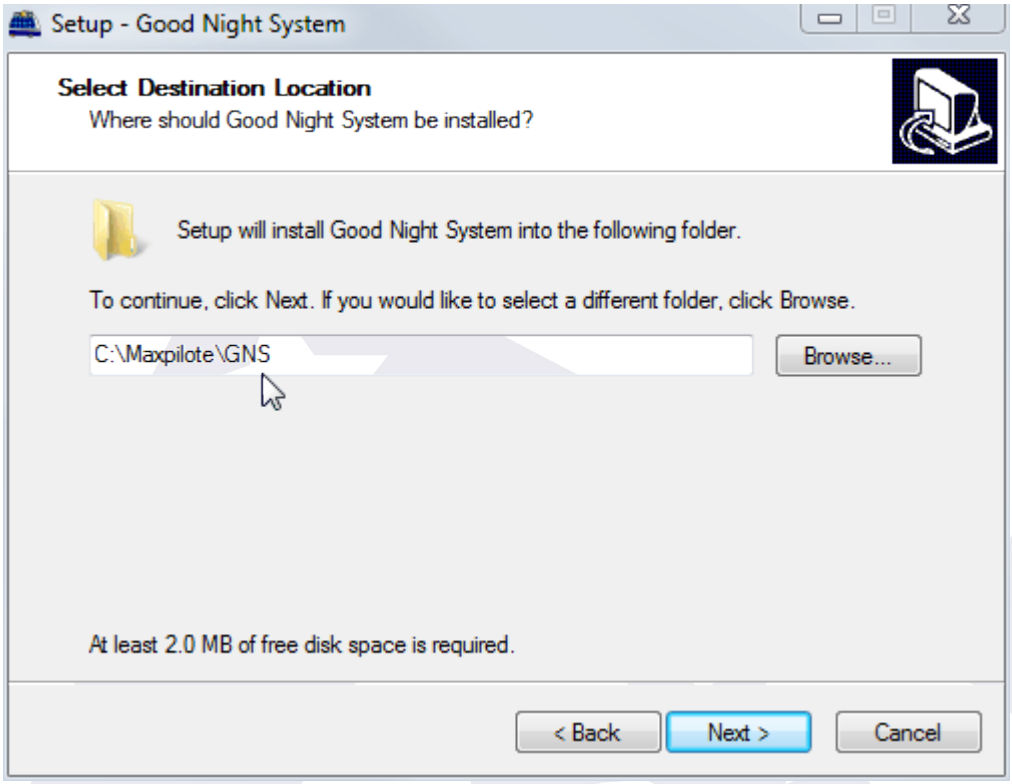
Maxpilote's author has already added native support for the Good Night System (*merci beaucoup, Laurent!*):



... while I've been unable to test it fully, I'm sure any possible issues will be addressed by him the great way Maxpilote users are used to.

Very important: for the GNS to work with Maxpilote, the software must be installed in the same folder as Maxpilote.

Please check as the installation program will try to install it under "Maxpilote\GNS" instead.

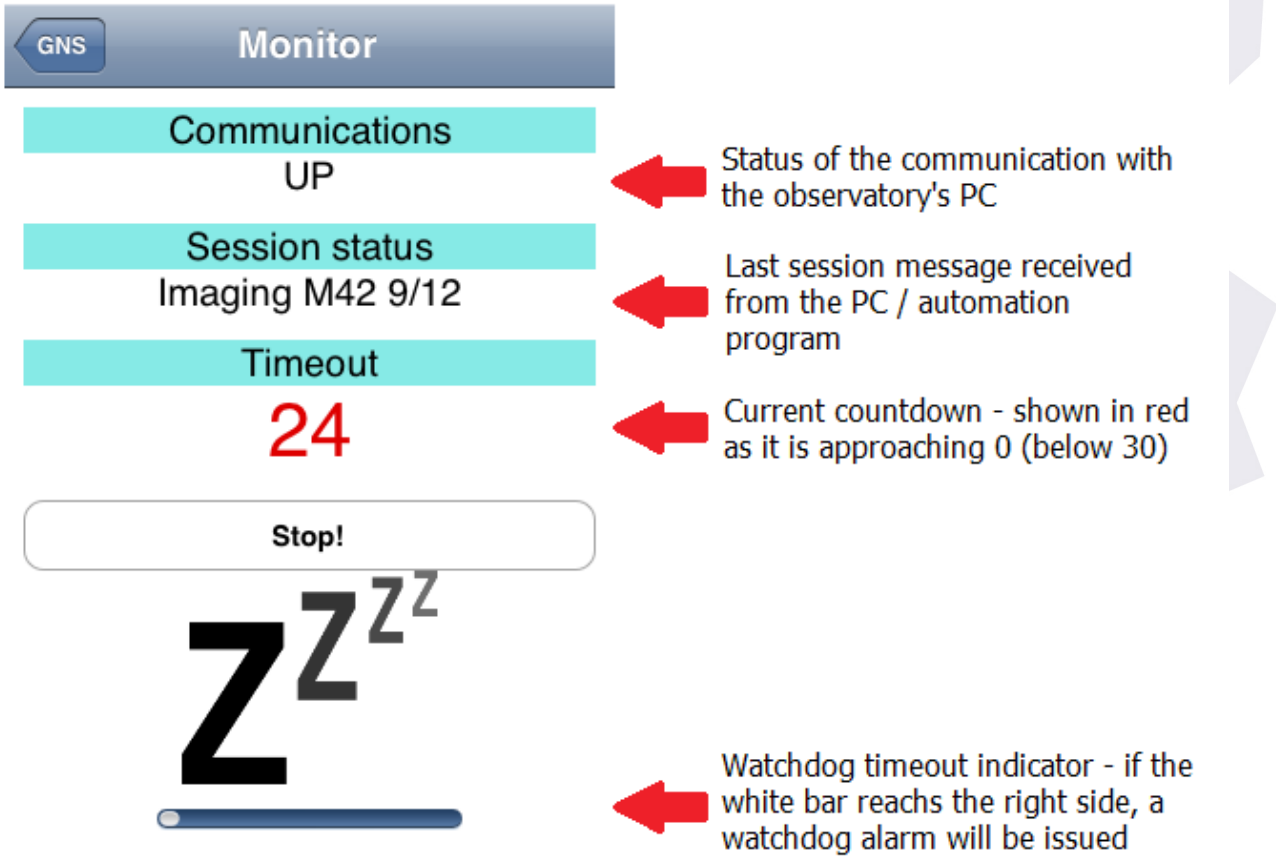


… you'll have to manually remove the trailing "\GNS" from the installation folder.

5. Important smartphone usage notes

– leave your smartphone fully charged or plugged to a suitable power supply – it won't warn you if the battery runs out!
– dim the screen to save battery
– check your sound settings if you are using the audible alarm
– **the app won't run in the background!** Communications are not reliable at all while in the background, so it is mandatory to leave the app running, the monitor screen visible. In fact, for iOS devices, if the app is "sent to the background", it will stop.

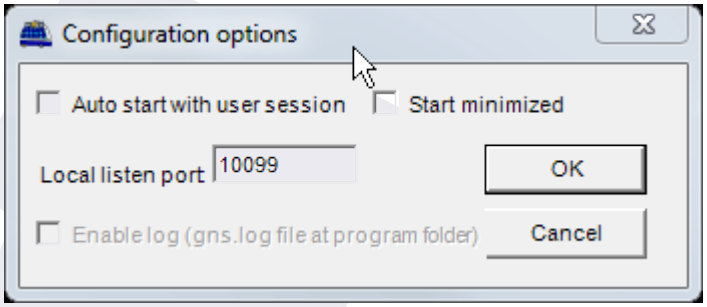
Just in case, here's an explanation of everything in the monitor screen:



The timeout shown in the smartphone will typically be a few seconds delayed compared with the one in the PC; this is normal.

6. Windows application options

Before going on, let's take a look at the few available options in the windows program:



Auto start with user session: if checked, the GNS program will be launched and will start accepting connections as soon as you log in windows. May be useful if you prefer to have your smartphone connected even before the real session has started.

Start minimized: if checked, the application will be "hidden" in the system tray, to avoid cluttering your desktop, every time it is launched. Of course you can make it visible again just clicking its icon in the tray.

Local listen port: this is the "slot" the program will be awaiting connections from. It must match the one selected in the smartphone app. We chose 10099 as ports above 10000 are freely available to programs; no problem in changing it if you have that port already in use, but please check the available ports in your smartphone app.

If you change the port, you'll have to restart the application for the change to take effect.

Last, you are not limited to the supplied scripts (even if everything can be done with just the "update.vbs" one); should you find it useful, just edit any of them with a suitable editor (the standard "notepad" will do nicely) to issue personalized messages easily.

7. Tight integration

Scripts are just a means of updating the GNS from the outside; this can be done directly from other programs, just calling the appropriate methods of the GNS "OWL" class (GNS.OWL).

There are two such methods:

NewMsg: takes a string argument, and just updates the displayed Status

NewTimeout: takes a long integer argument, and updates the timeout:

- if *timeout > 0*, the behavior is normal
- if *timeout = 0*, then an alarm is issued as soon as it is received by the smartphone
- if *timeout = -1*, then the count down is stopped, and the system is "waiting" (but the watchdog is still active!)
- if *timeout = -2*, then a switch off is performed (it is understood the end of the session has been reached).

There's no need to launch the user interface application (GNSUi.exe), it will be launched, if needed, upon creation of the GNS.OWL object.

I'm absolutely confident any programmer will find this explanation more than enough, but of course I'll be glad to help.

8. Pending features and wishlist

At the moment of this writing, I am aware of no errors, but already have a list of things to add in future updates; everybody is invited to contribute with ideas. My apologies for the folks of the beta testing team as most of their suggestions are still there:

PC:

– activate the local log, so one can make a forensic analysis if something goes wrong with the session.

Smartphone:

– if the session is switched off, then the smartphone can be allowed to sleep
– a "test sound" button in the configuration screen
– offer more sounds... not everybody likes the barn owl "bark".
– allow the user to set the volume in the configuration screen
– when the timeout reaches 0 and the alarm is fired, display the last status message.
– Add a timestamp of the last status message
– clear the counter when the status is set to "-1" (idle)

System / General:

– add another status message, so an additional program can check, for instance, the weather, roof status, mount parked status and report it to the smartphone.

9. Last notes

I'm confident you understand there's no warranty, whatsoever. Use the software at your own risk. If your expensive equipment gets damaged for any reason, do not blame us.

Developing the system has taken much longer than expected, for a variety of reasons, but it's been worth the effort.

Acknowledgments

Laurent Bourgon, for taking the time to add support for the **GNS** in his Maxpilote software.

Gerald Hitz, for writing the section on ACP integration.

John Smith, author of CCDAutoPilot, for his nice integration of the **GNS** in his software.

Matt Thomas, author of CCDCommander, for allowing me to post the messages about the system in his support list.

The **Mosync team** for their great cross-development tool and their continuous support.

… and of course everybody in the beta test team for their contributions and patience.

Revision history:

– 1.0 Original manual

– 1.0.0.1 More clear 32/64 bits explanations, more detailed Maxpilote installation, updated everything related to CCDAP v5. New "message.vbs" script.