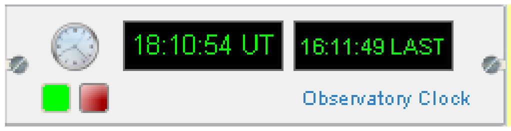


Almost everything you will do with an instrument will require knowledge of the time. Normally a software program will obtain Time from the real-time clock hardware on the host computer. This clock is maintained by a battery, and is usually set by the user, or perhaps adjusted now and again from the network. However, the observer will usually need the local time (or *wall time*), sidereal time, or maybe fixed points as a reference. It is easier if these moments in time are always available to all instruments to avoid recalculation, and perhaps a loss of synchronisation. The `ObservatoryClock` therefore provides a common Time service to *all* instruments. It is worth noting that the clock instrument you see in the rack on the left hand side in each of the Group tabs is exactly the *same* clock, so if you stop one clock, they all stop, and so on.

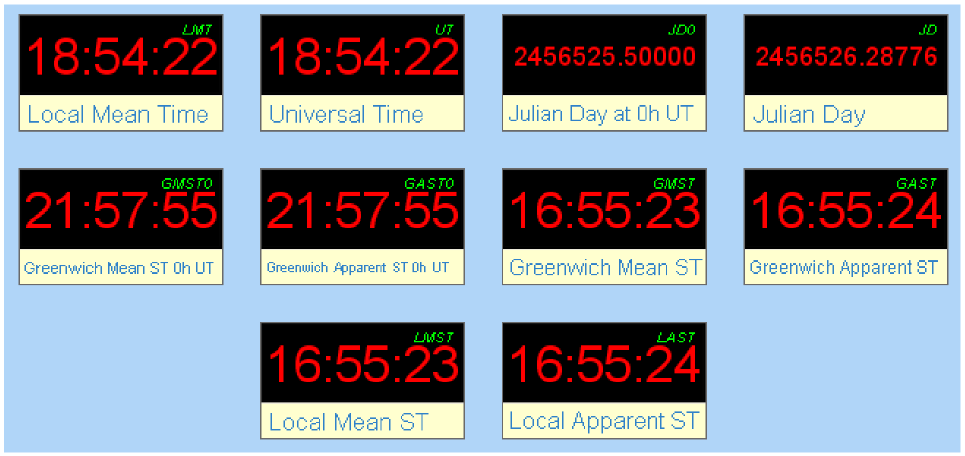
General User

Firstly, start the clock by clicking the small green button which appears on the clock's control panel in the rack of instruments. Most users will probably be quite satisfied using the default setup for the clock, which uses the host computer's hardware. The time may or may not be accurately synchronised with some reference Time. We've called this the **`SimplePlatformClock`**.

User views a Wall Clock

Here the user is looking at the Time on a clock on the wall. The control panel of the `ObservatoryClock` is set to show the time in this way. Simple isn't it? But sometimes things are not quite as they seem...

What other functions are available to the user in the Clock's interface?



LMT 18:54:22 Local Mean Time	UT 18:54:22 Universal Time	JD0 2456525.50000 Julian Day at 0h UT	JD 2456526.28776 Julian Day
GMST0 21:57:55 Greenwich Mean ST 0h UT	GAST0 21:57:55 Greenwich Apparent ST 0h UT	GMST 16:55:23 Greenwich Mean ST	GAST 16:55:24 Greenwich Apparent ST
	LMST 16:55:23 Local Mean ST	LAST 16:55:24 Local Apparent ST	

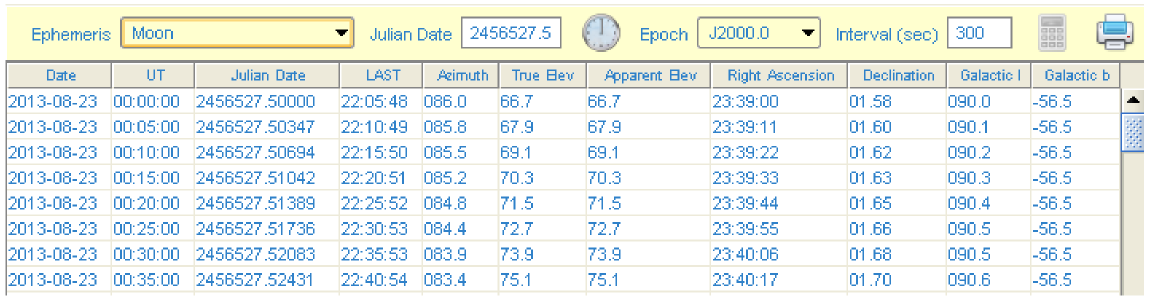
Time System Clocks

As soon as the clock is running, the Clocks Tab shows all of the *time systems* available, which are all calculated on the fly from the basic timer running in the computer's clock. The most useful will probably be `UniversalTime` (**UT**), and `LocalApparentSiderealTime` (**LAST**), which by default will appear on the control panel of the `ObservatoryClock`. If you prefer to see other times, then the panel may be configured to display any two of these time systems:

JD LMT UT GMST GAST LMST LAST

`JD0 GMST0 GAST0 will also display, but are not very useful!`

There are several other generic Tabs on the user interface (UI), but one which is more specific to the clock is the Ephemeris generator. Here you can select one of several target objects, and given a time step and a location, calculate the coordinates of the target as shown below. It is straightforward to change the list of ephemeris targets, to include a few of your own, or to remove those that are unwanted.

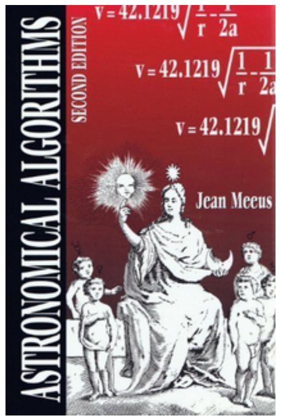


Ephemeris: Moon Julian Date: 2456527.5 Epoch: J2000.0 Interval (sec): 300

Date	UT	Julian Date	LAST	Azimuth	True Elev	Apparent Elev	Right Ascension	Declination	Galactic l	Galactic b
2013-08-23	00:00:00	2456527.50000	22:05:48	086.0	66.7	66.7	23:39:00	01.58	090.0	-56.5
2013-08-23	00:05:00	2456527.50347	22:10:49	085.8	67.9	67.9	23:39:11	01.60	090.1	-56.5
2013-08-23	00:10:00	2456527.50694	22:15:50	085.5	69.1	69.1	23:39:22	01.62	090.2	-56.5
2013-08-23	00:15:00	2456527.51042	22:20:51	085.2	70.3	70.3	23:39:33	01.63	090.3	-56.5
2013-08-23	00:20:00	2456527.51389	22:25:52	084.8	71.5	71.5	23:39:44	01.65	090.4	-56.5
2013-08-23	00:25:00	2456527.51736	22:30:53	084.4	72.7	72.7	23:39:55	01.66	090.5	-56.5
2013-08-23	00:30:00	2456527.52083	22:35:53	083.9	73.9	73.9	23:40:06	01.68	090.5	-56.5
2013-08-23	00:35:00	2456527.52431	22:40:54	083.4	75.1	75.1	23:40:17	01.70	090.6	-56.5

Ephemeris of the Moon

The above output shows the position of Earth's Moon every 300sec, for the Observatory location. This output may be exported or printed, so a variety of documents may be generated for use during an observation. Note that the clock **must** be running, and the Observatory Metadata **must** contain the Observatory's location (`Latitude, Longitude, HeightAboveSeaLevel`). If you used the installer to set up Starbase, then all of this information should already be present. If not, then use the Metadata Tab to import the Observatory metadata. This will give a location on the equator, and so if you select the Starmap Tab with these settings, you will see that the sky appears quite different.

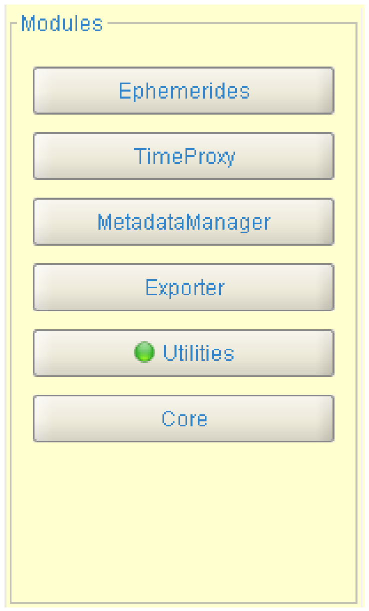


The various calculations used to generate the ephemerides are taken from this book, **Astronomical Algorithms**, by Jean Meeus, a classic in its field. I have faithfully followed the details in the book, but I cannot guarantee that the generated ephemerides are error free! This part of the software is particularly difficult to test, because of the many variations of input data (locations and times). Your feedback is much appreciated.

Starscript

By now you will have noticed the Commands Tab, which contains several control buttons, grouped into Modules, Commands and Parameters. All of the various functions of all of the instruments are controlled in this identical way. The *user interface metaphor* is much like a programmable oscilloscope or signal generator: the user selects the operation required and the values of any parameters, and then executes the command. Grouping the Commands into Modules simplifies the structure, and makes it easier to remember where to find specific commands. This hierarchical structure lends itself well to some kind of scripting or *macro* language, which we hope to implement one day. In the meantime, the full description of a Command is called its **Starscript**, in anticipation of such a language, structured as below:

```
Instrument.Module.Command(ParameterA, ParameterB)
```

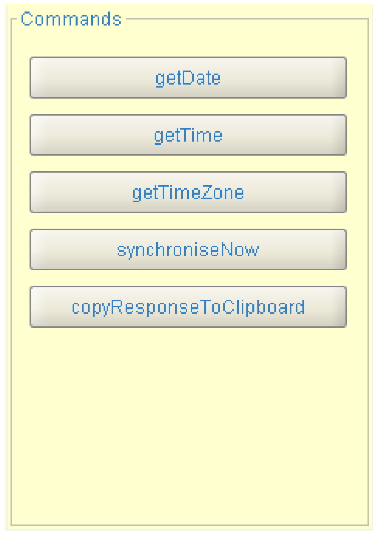


Observatory Clock Modules

Several of the clock Modules are common to all instruments, so we'll concentrate on the clock's features. The **`Ephemerides`** Module simply duplicates the toolbar functions of the Ephemeris Tab, so that in principle it is possible to generate an ephemeris programmatically. This may one day be useful for *e.g.* automatic control of antenna position, where a **Starscript** program generates the coordinates and moves the antenna.

The **`TimeProxy`** Module is under development, and is described in the Expert User section below. *Note* that if the clock is not set up to use a Time Proxy, then some command buttons will be greyed out (*i.e.* unavailable). We decided to leave the buttons visible, so that you become aware of the other features which can be enabled. Simply hiding the buttons would simplify the interface, but would probably generate more questions about missing functions.

The **`Utilities`** Commands appear as shown below:



Utilities Commands

These are very straightforward. For instance, if you need to get the Time (as opposed to just looking at the clock), then execute the following Starscript Command:

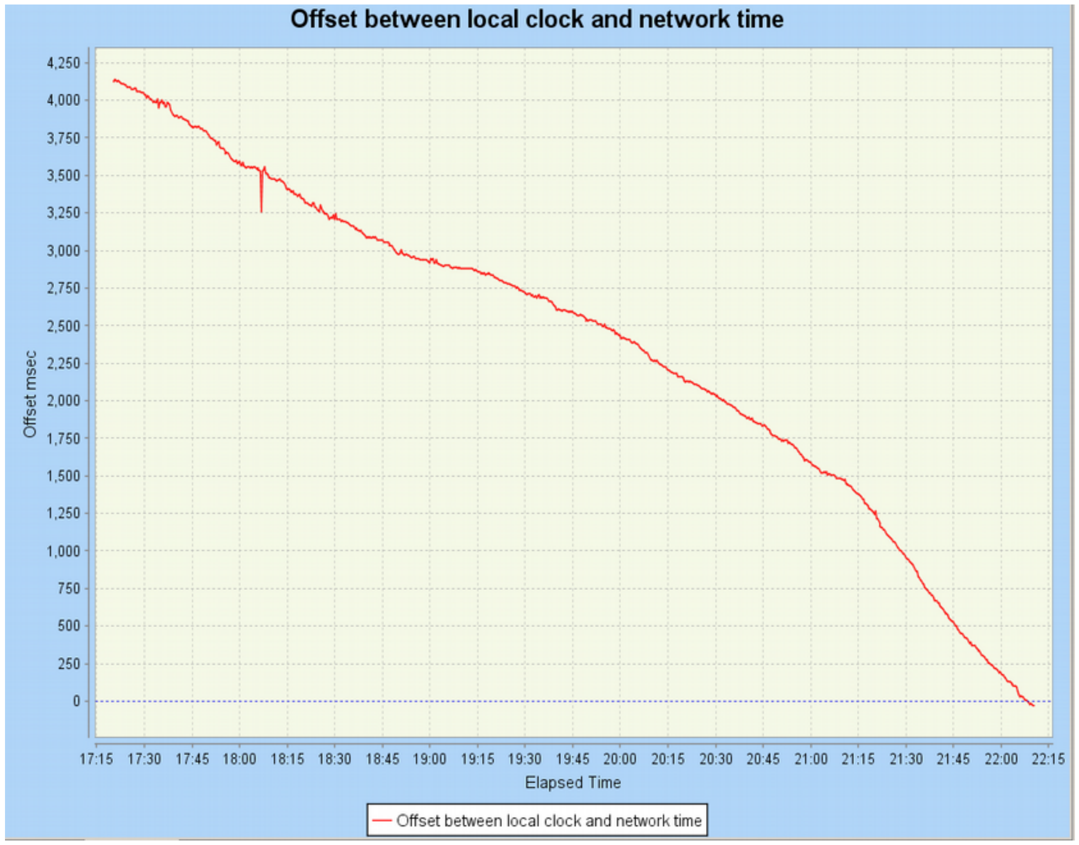
```
ObservatoryClock.Utilities.getTime()
```

Where the empty parentheses '**`()`**' show that no Parameters are required. The response to the Command will be returned on the Response tab, and also in the Command Log window. The Response may be copied to the clipboard, and then pasted into other applications. This operation may seem rather trivial at the moment, but it is the first step to putting into place the ability to execute Commands automatically, recover their responses, and feed some values into subsequent Commands, thus building complex control routines.

Expert User

Now is probably a good time to invest in a new pair of open-toed sandals, and to grow an unreasonable amount of facial hair...

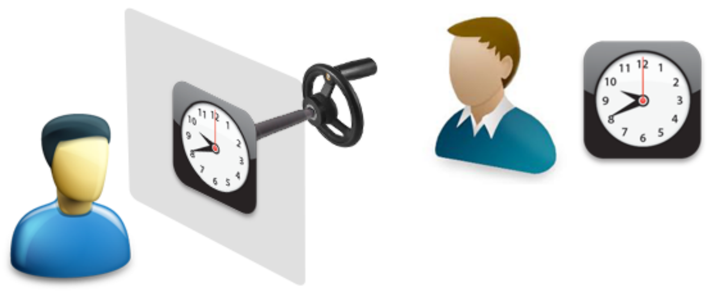
I mentioned earlier that the **SimplePlatformClock** 'may or may not' be accurately synchronised. What happens if you use the computer's clock just as it comes? Well unless you are very lucky, it will drift. The recording below shows how my computer clock drifted relative to time obtained from the network (via Network Time Protocol, or NTP). Clearly after a very short time the clock would become useless for anything other than the simplest observations. The amount and rate of drift will depend on many things, and if you try this experiment, your results may be very different.



Unadjusted Clock Drift vs. Time

So we need some way of ensuring that the `ObservatoryClock` presents an accurate time to all observatory instruments, but without asking for user intervention if possible. Some operating systems (OS) allow users to set up a synchronising process that regularly connects with an NTP server and adjusts the local clock. Some OS do not allow such changes to other than users with administrator privileges, which can sometimes cause problems. Also the method of synchronisation is restricted to that provided by the OS, *i.e.* NTP, or to custom external software.

The method finally chosen to overcome these restrictions and to allow complete control over the *clock steering algorithm* is call a Proxy Clock. The diagram below attempts to explain now this works. The user on the left sees the 'wall clock' time as before, and is unaware as to how the time has been generated. The users does not know, or need to know, what goes on behind the wall. The helpful, but rather controlling, character on the right has access to a master clock, and every now and again adjusts the time on the user's clock to match that of the master. So what the users thinks is a clock is nothing more than a simple indicator, driven from somewhere else. *Ceci n'est pas une horloge.*


User views Proxy Clock, set by a hidden process

This *abstraction* of the function of the clock is a common theme in programming. It allows the functionality to be changed at will (even at runtime) without changing the software that uses the service provided. So in this case, the user keeps calling **`getTime()`**, but the Time may be coming from somewhere else.

One of the simplest ways to obtain an accurate Time is via the Network Time Protocol, which accesses layers of time servers, synchronised to atomic clock standards. Starbase has an alternative clock, called the **`NTPSynchronisedProxyClock`** which regularly adjusts a **software** clock (as opposed to the computer's hardware clock) to keep in step with the NTP reference time. A simple algorithm attempts to work out the drift rate of the local clock, and adjusts the clock rate to try to stay synchronised with the NTP master. This should therefore remain accurate between synchronisation events. It is important to realise that it does not affect the host computer hardware clock in any way.

Supposing that the clock is set to use **`NTPSynchronisedProxyClock`** (see how this is done below), then it will indicate `00:00:00` until the following Command is executed:

```
ObservatoryClock.Utilities.synchroniseNow()
```

The user may then set up a suitable synchronisation period using:

```
ObservatoryClock.TimeProxy.setSynchronisePeriod(?)
```

where **'?'** represents the requested sync period in seconds.

Then start the automatically repeated synchronisation using:

```
ObservatoryClock.TimeProxy.synchroniseTimeProxy(true, true)
```

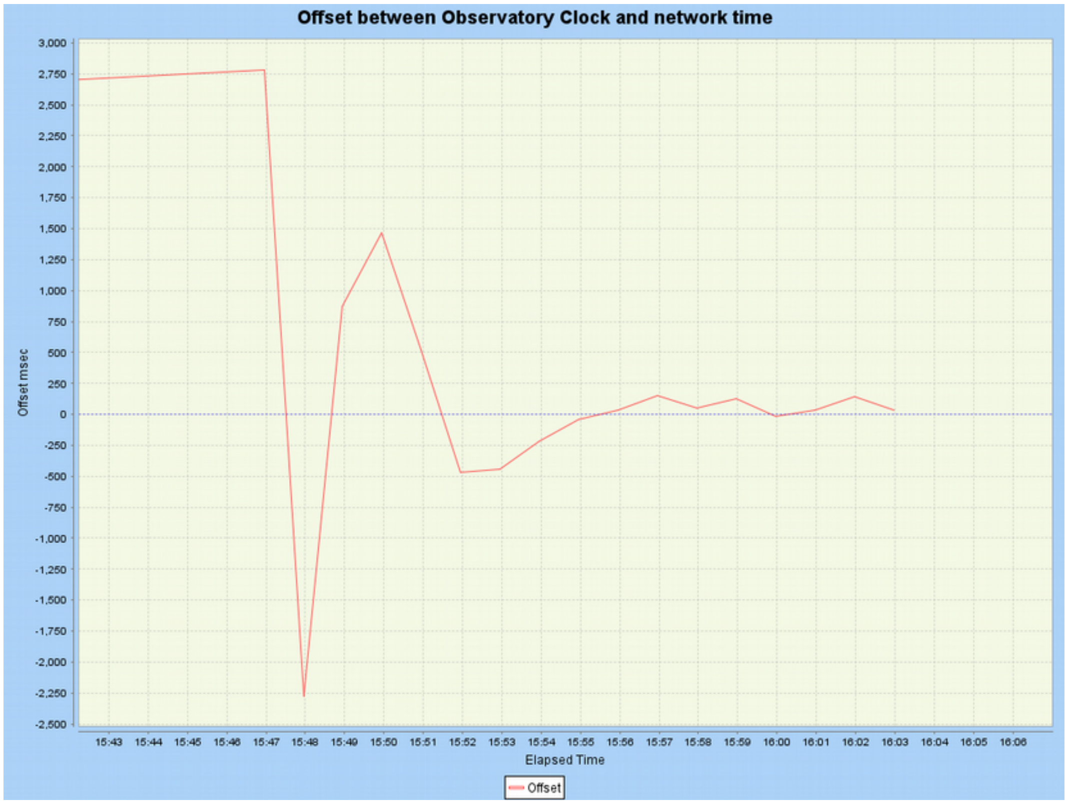
Watch the Synchroniser Chart Tab to see how accurately the clock is running (or not). If you have enabled debug messages, you will see something like this in the console window:

```
[target=clock] [action=synchronise] [offset=-127 msec] [driftrate=-7.52 msec/sec] [timestep=251 msec]
```

This software is very much under development, and could be improved considerably with a bit more effort. One improvement would be to expose the tuning parameters of the clock steering algorithm via Command settings, so that the users can experiment to fine tune the clock. All in good time!

Start here http://www.ntp.org/ for a discussion of NTP and many useful links to other resources. Please remember not to hit an NTP server too frequently! This would be regarded as bad 'netiquette'. It would be easy to create a clock synchronised every second by NTP, but you would soon find that it wouldn't be a popular implementation.

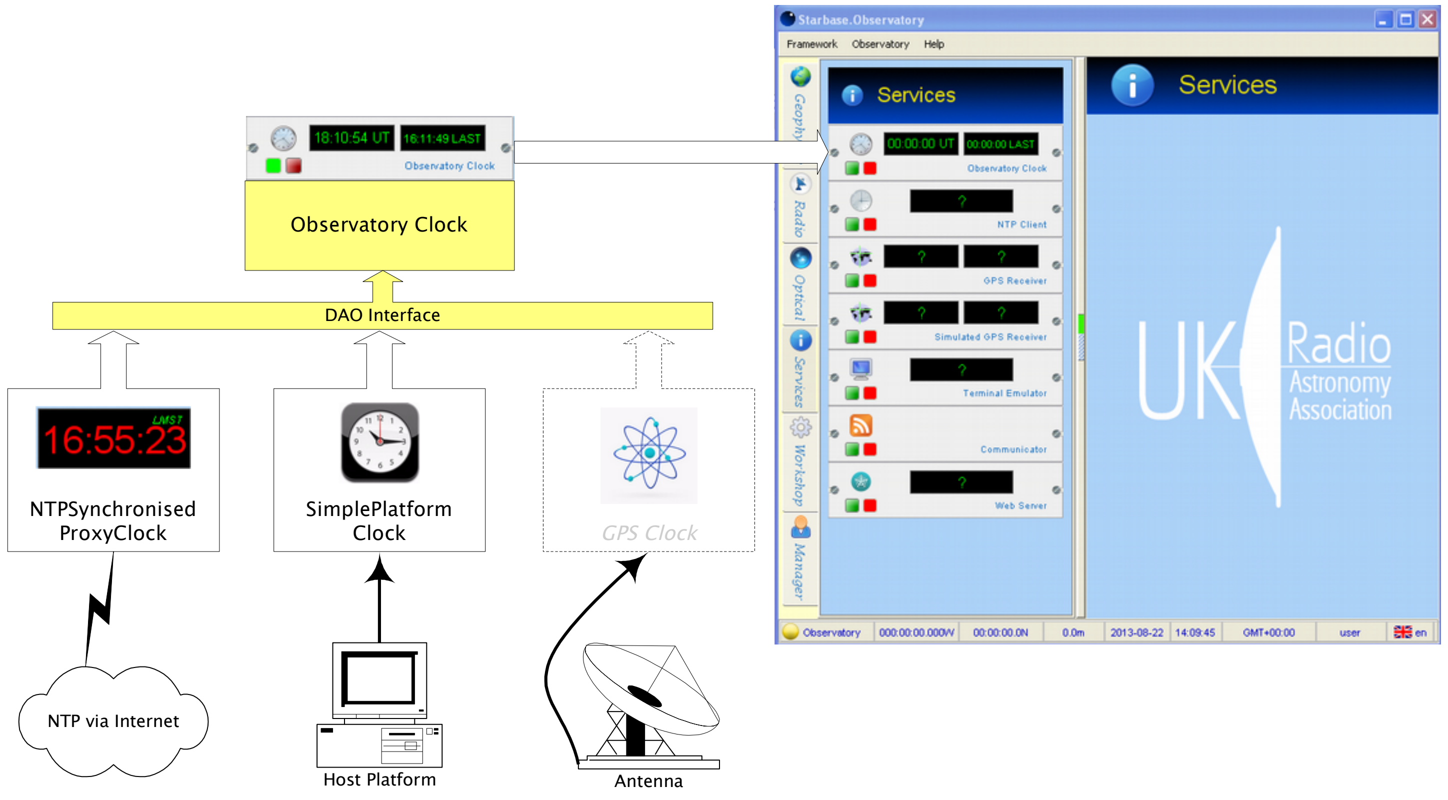
The results of some initial tests of the algorithm are shown below. The clock starts up a few seconds in error, but rapidly settles to within about 100msec. This is almost certainly good enough for our uses, but more accuracy might be obtainable.



Clock Drift adjusted using NTP

So, suppose we did need extra accuracy, or just wanted to experiment with other time standards? The clocks we have mentioned, the **`SimplePlatformClock`** and the **`NTPSynchronisedProxyClock`** are written as Java *Data Access Objects* (DAO). This means that the software obtains the data it needs from this object, and that object alone is responsible for gathering the information (in this case the Time) from the outside world.

Creating another type of clock simply means writing a DAO to obtain data from a different system, for example as shown in the diagram above, from a GPS synchronised source. The architecture below allows for any number of different types of clock, but as far as the user's software is concerned, it still calls **`getTime()`**.



Choice of Source of Time via DAO Selection

So by now you will be asking, '*How* do I select which source of Time to use'? With the answer comes a warning that it may mean delving into new territories. Since Starbase is a *data driven* application, *i.e.* it is configured entirely separately from the software itself, it is necessary to edit a configuration file (plain text). The configuration is encoded into the formal structure of **`XML`**, and is done this way to minimise the risk of errors and ambiguity. So to choose the Time source, we must specify the DAO to use for the Instrument (described above), and edit the `ObservatoryClock-instrument.xml` file to select one of the following two entries:

either

```
<!-- SimplePlatformClockDAO just uses the PC platform clock -->

<DaoClassname>
org.lmn.fc.frameworks.starbase.plugins.observatory.ui.instruments.impl.clock.dao.SimplePlatformClockDAO
</DaoClassname>
```

or

```
<!-- NTPSynchronisedProxyClockDAO runs independently of the PC platform clock, synchronised by NTP -->
<DaoClassname>
org.lmn.fc.frameworks.starbase.plugins.observatory.ui.instruments.impl.clock.dao.NTPSynchronisedProxyClockDAO
</DaoClassname>
```

Later developments may include other DAOs, such as a GPS synchronised source, as discussed earlier. Once you have learnt Java, it is not too hard to write your own components for this system, because everything is very modular and compartmentalised. A new DAO to derive the source of Time from your own project is quite feasible.

Laurence Newell
2013 November