STARBASE ~ A Software Observatory

Laurence Newell

Introduction

This paper is an overview of a general-purpose Java software package called *Starbase*, which was written by myself, and further developed and distributed by the UK Radio Astronomy Association (UKRAA) [1] on behalf of the Radio Astronomy Group (RAG) [2] of the British Astronomical Association (BAA) [3]. It is intended for the collection, analysis and storage of data from multiple data sources, specifically (but not exclusively) for astronomical applications. As with most technical papers, I use a lot of abbreviations and acronyms. To help the reader, I have provided a Glossary at the end.

The current release of the software contains an Observatory, with facilities to collect data from local instruments or remotely over the Internet. Planned future developments include a data repository, and an analysis module. Starbase has an unusual and highly-flexible architecture, is platform-independent and will run under MS Windows, Linux, or MacOS.

I don't claim to be an observational astronomer; my interests lie mostly with technical problem-solving in data collection and analysis, with the associated control systems, so this paper is mostly about software rather than radio astronomy. I am hoping that you find something useful in what I have created.

What does it Do?

The Starbase Observatory connects to a variety of instruments, presenting their data in consistent graphical and tabular forms, and allowing storage and retrieval from the local file system. The user interface *metaphor* is that of a programmable oscilloscope or signal generator, and so should be familiar to most users of radio equipment. The command interface is specifically intended to be scriptable, to facilitate building macros (“batch files”), or running unattended: the macro facility is under development, when completed, custom interfaces will be possible for each instrument. The Observatory communicates with hardware instruments either via the serial port with RS-232 or RS-485, or via an Ethernet connection (e.g., FTP, HTTP). USB interfaces are more difficult, but are being explored (e.g., the FUNcube dongle [4]). Software instruments entirely resident on the host computer are also supported.

Starbase has a unique design, with an extensible *fractal* architecture, which is fully configurable by editing XML files that follow a strict structure (or *schema* [5]). So for instance any new instrument may be added to the Observatory but it must first be described in an XML file. Some additional Java software may also be necessary, depending on the functionality required.

The Observatory currently connects to external data loggers, a VLF receiver, magnetometer, SpectraCyber (local and networked), GOES server, FTP servers, NTP services, GPS receivers, a terminal emulator, and an Ethernet webcam. It can import and export data in most combinations of XML, comma-separated value (CSV), tab-separated value (TSV), RadioSky Pipe (RSP), GOES data, RSS feeds and even Twitter. Images in JPG and PNG are supported, and FITS is *in embryo*. There is a built in webserver, a user-configurable astronomical ephemeris, and a simple time proxy for accurate timing.

Where did it come From?

I have been interested in radio astronomy for many years, mainly as an excuse to build various antenna systems and their control electronics, and to play with signal processing. My university third-year project to build a phase-switched interferometer at 151 MHz in the dark days of chart recorders was the starting point of many such experiments. My research project in high-power lasers and plasma diagnostics introduced me to many new electronic and mechanical skills and measurement techniques. A career as a physicist and electronics engineer with British Telecommunications (BT) [6] followed, and I later moved from electronics into software. I began to realize that there was much I could do with a computer to control my telescopes and analyse their signals. Luckily, part of my work involved the design and manufacture of microprocessor control systems, and I began to specialize in FORTH [7]. I also used C and assembler with various processors, and developed control applications for my personal interests. I got as far as a stepping motor controller for a dish mount, and realized that

cross-target development in assembler was *hard*, and although FORTH was fun, it did not come as easily as C, for instance.

In the early 2000s, my work moved more into development for the Internet, and eventually to Java and databases for large commercial systems (J2EE, JSP, SQL). I found web development to be unsatisfying; the feeling I had was of a thirty-year-old technology being tortured to do something for which it was never intended. I wanted to do something more interesting, preferably involving radio astronomy and control systems. In my spare (!) time, I was part of a small team that reformed the dormant Radio Astronomy Group (RAG) of the British Astronomical Association (BAA). We realized early on that we needed some kind of common project to enthuse our membership, and came up with the idea of the *plug and play observatory* [8], since we had received many enquiries from people who wanted to get into RA, but who had little or no electronics or software experience.

So began the Starbase project, where we aimed to produce some simple hardware and software systems, which were to be made available via the BAA. The next couple of years were filled with many meetings and discussions about the design of an architecture which would meet the plug and play requirement, and yet remain simple for the user to operate. We put a lot of effort into the design of hardware modules which could be mixed and matched, and provide data to the host computer to identify their specific configuration. We also designed and implemented a custom communications protocol which could be used to interface microcontrollers or PCs to the host software. This became known as *Staribus* (***Star***base ***I***nstrumentation ***Bus***) and is loosely based on Modbus [9] [10] [11]. This protocol has stood the test of time, and has been successfully implemented for RS232/485 and Ethernet.

The software itself had several false starts; we soon realized that to make things simple for the user meant that things were very difficult for the developer. Events were however to take a helpful turn - in about 2005 a friend contracted me to develop a race results system for long-distance endurance horse racing [12]. I decided to try my new found skill of Java programming, and eventually came up with a multi-PC system with multi-user access, a central MySQL database, and interfaces to many RS232 event timing units [13]. All of this experience was invaluable to my 'other job' as the BAA RAG software architect, and the software that became known as Starbase is in fact based on a race management system, which is a good example of one of the Design Goals outlined below.

The end result of all of this work has been a set of high-quality hardware modules, software and documentation which are available from The UKRAA, the trading arm of the BAA RAG. UKRAA is a not for profit charity, and is staffed by volunteers. The Starbase software is Open Source, provided free of charge. It is licensed under the GNU General Public Licence (GPL) [14]. The project is very much a work in progress, but is becoming easier now that more people are involved in its evolution and testing.

Why is it like it Is?

I knew from the beginning that I was setting out on a personal journey, a Grand Design to create some software as I felt it *should* be done, specifically to meet my own needs and my own ideas of 'perfection'. This means I have ended up with something a little bit different – you may like the approach, you may not. I knew also that it was going to be very much a geek's toolset, something where I wasn't expecting to worry about catering for the beginner or the inexperienced. Much of the software I had used irritated me, perhaps in the over-use of dialog boxes, popping up when least expected, or poor error reporting (perhaps none at all). Cluttered menus, where the function needed was buried deep; or maybe the dreaded 'hide the menus you don't use much' option, multiple ways of finding information, and so on. There were many aspects that I felt I could improve, and make consistent, at least in a way that made sense to myself.

As the software matured, and my involvement with RAG grew, I changed course slightly to improve the usability for novice users, and this in turn helped with improvements in the inner design. It's still not something you could use without a little training, or reading of the (skeletal) manual, but the concepts are very consistent and logical. It has been a very interesting and instructive path, I have learned a lot, and have ended up with a flexible tool which provides many off-the-shelf solutions, but it is also a good prototyping system.

Although the design goals for this project are fairly easy to state now, they have of course evolved over many years; this list below is what I would have written, had I known then what I know now. With hindsight, and now knowing the effort required to achieve these goals, I might have taken a very different (and easier) path, but to paraphrase JFK, I chose to do the other things, not because they are easy, but because they are hard.

No Assumptions: *There must be no assumptions made anywhere in the software about its purpose.*

Another way to describe this might be to say that it must be a true *generic solution* (or if you prefer, an *impossible* solution...). This route ensures re-use of software modules, and makes it easier to add new functionality without changing the underlying architecture. Of course it does mean that there is pressure on the designer to make sure the architecture is correct before moving too far along the implementation path. I did have to make one or two changes as time went on, but this goal has been getting closer all the time. The really positive thing is that it has forced me to *generify* or *abstract* the functionality, often revealing assumptions that were not initially recognized as such. The end result is more stable, more testable, and easier to maintain and to modify. This is definitely the hardest goal to explain – I am often asked “why can't you?” and the response is usually “how do I tell the software what you want to do in a *generic* way?”.

Platform Independence: *The software must work on all of the three popular platforms (i.e., Windows, Linux and Apple Macintosh).*

This decision narrowed down considerably the choice of development tools, and as stated earlier, the language chosen was Java. I have found this to be a very productive and creative environment, and the thriving open source community is an invaluable library without which I would still be writing the infrastructure. See for instance the Apache Software Foundation [15], but there are many others. I was also uncomfortable with Microsoft's domination of some areas of software development, and the freedom of Java I saw as a definite positive. The huge user base of Java provides all of the support and tools that a developer could ever need, with the distinct advantage that much of it is free, and not tied to any one particular supplier.

Data-Driven: *All aspects of the software's behaviour must be controlled by external data.*

This is maybe the easiest goal to state, but the hardest to achieve. The developer has to be *very* disciplined to ensure that there is no 'hardcoding' of data, and that *everything* is exposed to the outside world in a way that could in principle be changed by the user if necessary. This goal is probably the source of the decision to use XML files as the configuration mechanism – editing a plain text file can change the system behavior considerably, but in a controllable, structured way, which helps both the user and the developer. In practice I probably didn't achieve this goal to my own satisfaction, since several aspects remain fixed, or hidden from the user, for simplicity.

Extensible Architecture: *It must be possible to extend the functionality of the software without changing the underlying framework.*

In my career as a software engineer I had come across far too many systems which were being 'maintained' years after their inception, usually by bolting on things the original designers had never envisaged, and the act of 'bolting on' was making the design more unrecognizable with each new feature. Starbase took a different approach: a relatively simple framework (or *container*) provides services and infrastructure for a set of *plug-ins*. These plug-ins provide the real functionality of the software, and can be attached where they are needed. By analogy, think of the drop down menus of a 'normal' program. It is possible for each menu item to have another child menu, and so on. In the same way, a Starbase plug-in can have other plug-ins to provide more detailed functions. Other software packages do have a pluggable architecture: Starbase is unusual in that *all* aspects of the functionality are pluggable, even the user interface is a Framework plug-in.

This structure extends also to the attached hardware. Starbase communicates with external controllers, data loggers, servers and so on. Each controller could have hardware plug-ins, so for instance a data logger may have a receiver plug-in. Provided that these modules are compliant with the appropriate protocols, Starbase sees the pair as an intelligent peripheral, a receiver with a command set specific to

the combination of the controller and receiver. The user interface automatically presents this merged command set, and the user appears to be controlling a single entity.

Starbase has a *fractal* tree-like architecture, meaning that however closely you look, there may be more detail revealed, but which has the same form as the higher levels ("self-similarity" [16]). Each node of the tree is an item of hardware and/or software, containing a description of its purpose, which can be read by the higher levels. Each level can have as many plug-ins (hardware or software) as required, and so on to any depth. Each plug-in can be distributed as a separate entity (a Java JAR file), allowing for easier field upgrades. The Observatory is therefore a plug-in to the Framework, and the Instruments are plug-ins to the Observatory. Some Instruments have pluggable software components, for instance the clock can obtain a source of Time in several ways. Some Instruments have hardware plug-ins, for instance a receiver module, or a magnetometer.

Even the name *Starbase* is supposed to evoke the idea of a common meeting point, where interfaces of all kinds come together to exchange information, much like the docking ring on the Star Trek Deep Space Nine station, where different craft can plug into a standard airlock configuration.

Common User Interface: *The software components must have a common user interface (CUI).*

All modern graphical user interfaces have a common user interface. In other words, they have application windows, which have menu bars, resizing tabs, toolbars and so on: every application presents the user with a familiar set of navigation aids. It was not always like this. It made sense to maintain this level of consistency within the application. Specifically for the Observatory plug-in development, I wanted to ensure that all instruments had the same user interface, the idea being that once the user had learned one system, then all others would be accessible, or at least in part. As a simple real world example, consider a telephone. Although they come in many different shapes and sizes, the same interface is used, and because you know what a telephone *should* do (accept numbers and make a call), you can easily work out the details for a specific unit. Clearly trying to find a CUI for the instruments which would work equally well on a simple data logger, a web server, a clock or a radio receiver was difficult, and this goal did cause some problems in my settling on an implementation; this is discussed further in a later section.

Standards Compliant: *The software must be designed to adhere to all applicable international standards wherever possible.*

So for instance, use the ISO standard for dates and times, rather than confusing culturally-specific formats. [17] Also, use the *Système International d'Unités* (SI, or Metric) system for all dimensional units [18], and so on. The anecdote I recall here by way of justification is that of the Mars probe that went AWOL because one software module used SI units and one module used Imperial units [19]; a guidance command was misinterpreted. Imagine the embarrassment of having to explain to the taxpayer the huge loss because no-one had enforced proper standards across the whole project. I sometimes think that if we are ever visited by aliens, one of their engineers might exclaim "you have *how* many ways to measure a bolt?". The closest I will ever come to the Imperial measurement system is to use an icon of a 19 inch rack panel to represent an Instrument control panel!

Open Source: *The software source code must be open and free to all.*

I did not start out with the intention of making an Open Source [20] project, in fact very much the opposite, because I felt that the large investment of time and thought was sufficiently valuable to protect. I suppose that I hoped one day to make some sort of income from this work, but I found that working alone on such an ambitious task was testing my stamina a little too much, and it became more rewarding to meet user's needs rather than to (potentially) accumulate money. I was persuaded (eventually) to open up the project to a wider audience; this has turned out to have several benefits, in particular it gave BAA RAG some confidence in the future of the project if I should choose to do other work, or fell victim to the Red Bus Syndrome. It has allowed other designers and coders to properly peer review my work, and to contribute to its evolution. Also, it opened up a vast library of Open Source software, which could be freely incorporated, and has dramatically reduced the development times for what has become a very complex undertaking. Starbase now probably has more than 150,000 lines of Java code, excluding the dozens of libraries, and although this metric is

notoriously unreliable as a way of measuring the real size or worth of a project, it does give a feel for the difficulties faced by the developer(s) [21].

Now that others have joined the development team, we have found it useful to employ some modern technologies to help us manage the project in a cooperative environment. For instance, the source code is held in a configuration management system called Subversion (SVN) [22], itself free and open source: anyone may browse the Starbase source code on the Internet [23]. To keep track of bugs and request for changes we use the popular MantisBT bug tracker [24]. This list is also publically visible on the web, so that users can contribute their ideas. We now have a flexible way of working, which is similar to Agile programming [25], where we are motivated by customer's feedback, regular bug-fixing releases, technical interest and associated outreach activities, rather than by commercial gain.

> **Comprehensive Metadata Support:** *All data passing through the system must be accompanied by structured and consistent metadata.*

Metadata - from the Greek: μετά = "after", "beyond", "with", "adjacent", "self", in this context, *data describing the data themselves*. [26]

This goal is definitely one that evolved over the years, rather than being clear from the start, but it has become one of the unique selling points of Starbase. Another quick anecdote to show why this is important, and what I mean in more detail: I was given the entire life's work of an amateur radio astronomer, on dozens of paper chart rolls, with the hope that I “could do something with them”. It turned out that the charts were literally only recordings showing varying signal levels as a function of time – were they radio signals, or the temperature of the lab, or did someone just jolt the recorder? There were absolutely no indications of the context of the recordings, or even the subject matter. So, sadly the complete collection is effectively useless. Had the charts been annotated with, for example, the source under observation, a calibrated scale, the date, the name of the observer, and so on, then something useful or interesting may have been extracted. They were missing their metadata.

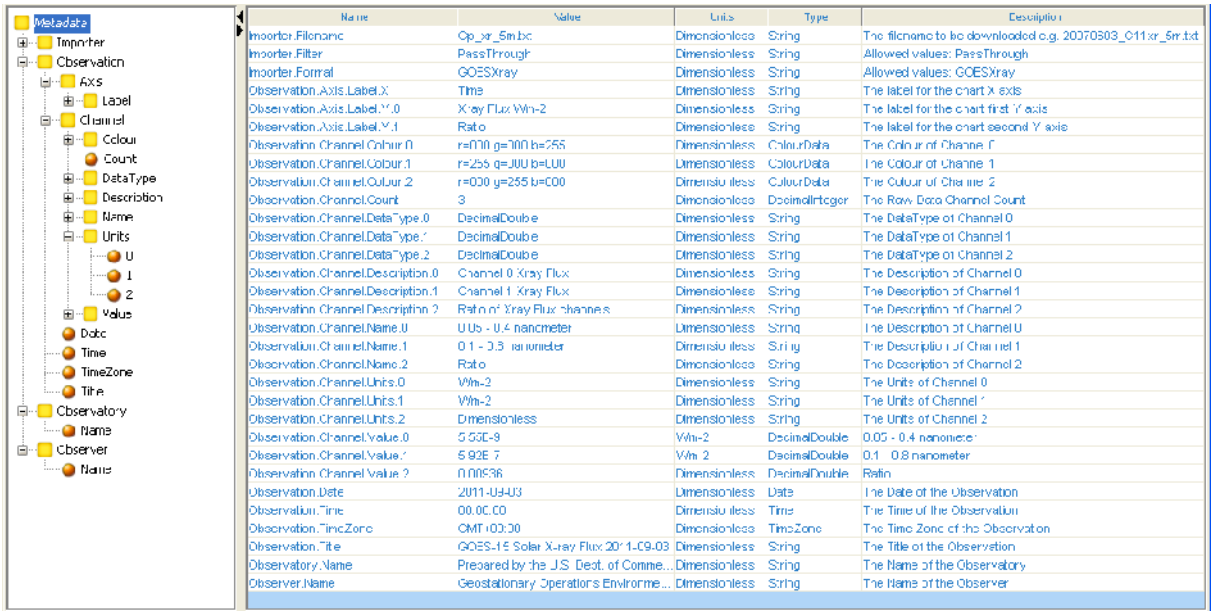
This experience reinforced my belief that this is one area which is often overlooked by the amateur observer (and perhaps even some professionals), and yet is relatively easy to implement in a useful way. The simple reason behind this design goal is to make sure that those observations I have carefully made are still understandable in the years to come, by others who have no knowledge about how the observations were made, or by whom. I needed to come up with a set of metadata which would completely describe every aspect of an observation, and implement it in a rigorous way which did not allow incorrect or invalid data to creep in to the record.

Starbase defines a simple (extensible) *Metadata Dictionary*, which is a set of predefined metadata items, which can be attached to the observational data. You may think that “surely this has been done before, why reinvent the wheel?”. Well yes, there are several wheels out there, but in my opinion (please prove me wrong) they are all not quite round enough or robust enough to continue rolling in reliable ways, as I hope to illustrate. Another related factor in this problem is the choice of file format in which to store the data. The format must of course also be able to store the metadata, otherwise each record would have at least two files, which is inconvenient and error-prone. Some formats do support metadata (e.g., FITS [27] and RSP [28]). For instance, FITS does have standard dictionaries for keywords [29] [30], although the names are simple, non-hierarchical, and do not appear to be derived from a consistent data model. One common problem is that different users often have different names for the same items, and perhaps different units and data types. For instance, a Time Zone may be described textually 'GMT+01:30' or as the number of hours offset from GMT, as '1.500', or in a culturally-specific form, such as 'CST' – is that Central Standard Time, or China Standard Time?. If different styles are used, how does the metadata *consumer* know what was intended by the metadata *producer*? Some file formats are not obvious candidates, but can be modified (e.g., CSV and TSV). For simplicity during development and testing, I also designed a new XML file format which fully captured the metadata (and the data), and translated easily into the Java data structures in the software. Starbase metadata relate to a comprehensive (but extensible) data model based on, for example, Observatory, Observer, Observation, Instrument, and so on.

I am not arguing that there is a 'best' file format, just that there should be a way of consistently and accurately recording the data *and* metadata. There are many tools for translating data into other

formats, but if the data are not there in the first place, they cannot be translated. If you are still doubtful about the necessity of the rigour of this approach, then try this simple test: take a data file which also contains some metadata. Import it into some software which recognizes the format, and then export in a *different* format. Then re-import back into the software, and export again as the *original* format. Are the two original format files identical? Do the files contain identical metadata? I doubt it. This is similar to the data corruption when using, for example, the JPG image file format, it is a 'lossy' process, whereas something like PNG is a bit-for-bit copy on each pass. Metadata distortion during repeated import-export iterations is digitally-enhanced Chinese Whispers, and if we really are going to make some advances in this area, we do need some reinforcements [31].

The screenshot below of the Metadata Explorer tab in Starbase should explain how the metadata are structured, and the ease with which they can be navigated, examined and edited.



Starbase Metadata Explorer tab (data from GOES satellite)

Are we on the same Wavelength?
Having read this far, you have probably formed some expectations of what Starbase is, and what it is not, and just maybe, what it could do for you. It might be worth having a quick reality check that your understanding of what I am offering will actually do what you expect.

Starbase is intended to be a general-purpose tool for the control and interrogation of a wide variety of 'instruments', which in this context could mean local hardware items, local software (built-in or third-party), remote file servers, and so on. Often these instruments will be used to gather data, and the aim is to provide a 'one stop shop' for the whole operation of data gathering, processing, and saving or publishing, all in the one application, all cross-platform, and open to all.

The command structure makes the distinction between the gathering of data directly from hardware sources, (by using `DataCapture` commands) or by reading ready-prepared local or remote data files (using `Importer` commands). The data collected are transformed into a common internal format; the *RawData* and *ProcessedData* (filtered) tabs show the numerical data, the *Chart* tab shows a configurable chart of all data channels. The `Exporter` commands can write a choice of popular file formats, from the content of *any* of the tabs. Later work should enable the publication of data and images directly to servers, perhaps with a `Publisher` module.

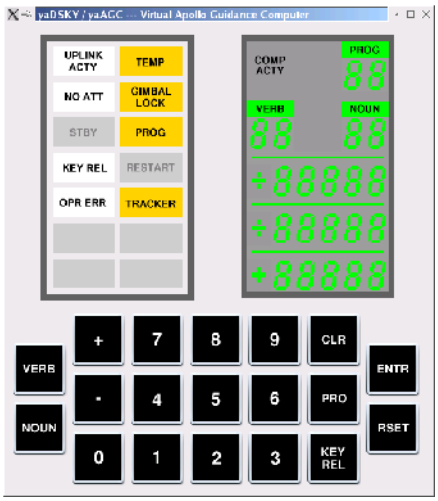
New instruments or data sources are fairly straightforward to add if they use one of the interfaces or protocols currently supported (e.g., file, FTP, HTTP, NTP, NMEA, RS232, Ethernet). Sometimes it is necessary to write more Java code to massage the commands or data into a suitable form, but this is my *raison d'être*, so just let me know what data source you'd like to use. We also have lots of ideas of useful things to add as controllers, such as antenna positioners, shaft encoders, receiver signal processing, and so on. Software Defined Radio (SDR) is another interest, and I am looking forward to the results of SARA's RASDR initiatives in this area. [32]

The Starbase software is **not** intended to replace the functionality of a realtime data logger, since the actions of repetitively retrieving data samples and displaying the results, perhaps from several instruments simultaneously, can be very processor intensive (remember that the number of instruments is essentially unlimited). If possible, it is usually better to log offline with external hardware, and download the data in one operation; this is the approach used with the controllers available from UKRAA. This method has the advantage that the host computer does not have to be running continuously. Alternatively, logging software such as Radio-SkyPipe [28], SpectraVue [33] or Spectrum Lab [34] can provide data files as sources almost in real-time, thus offloading much of the work to a helper application (provided that the platform-specific restrictions are not a problem).

Starbase is nowhere near as powerful in data analysis as, for example, Microsoft Excel, and it does not yet support scriptable macros, but both of these features are planned. Your suggestions for other features are most welcome.

How do I use It?
The command interface of Starbase was inspired by that used on test equipment from the likes of Hewlett Packard and Marconi. These pieces of equipment gained a reputation for being very easy to learn to use, and seemed a good model for the kind of interface required for Starbase.

Recall that the design goal was to build an interface which was conceptually the same regardless of the data source, so all 'command sets' of any kind of instrument are mapped to the same user interface. This would mean that once the users have learned how to operate one instrument, then they have essentially learned them all. The type of interface eventually chosen is known as 'noun-verb', where the *noun* is the target of the command, and the *verb* is the action – the user selects the item on which to operate, and then chooses the action to perform. [35] This simple structure lends itself naturally to Object Oriented (OO) programming, where *objects* (the instrument nouns) have *methods* (the command verbs) which operate upon them. An interesting example of this type of interface was used on the display unit (DSKY) for the Apollo Guidance Computer, shown below:



The Apollo Guidance Computer DSKY

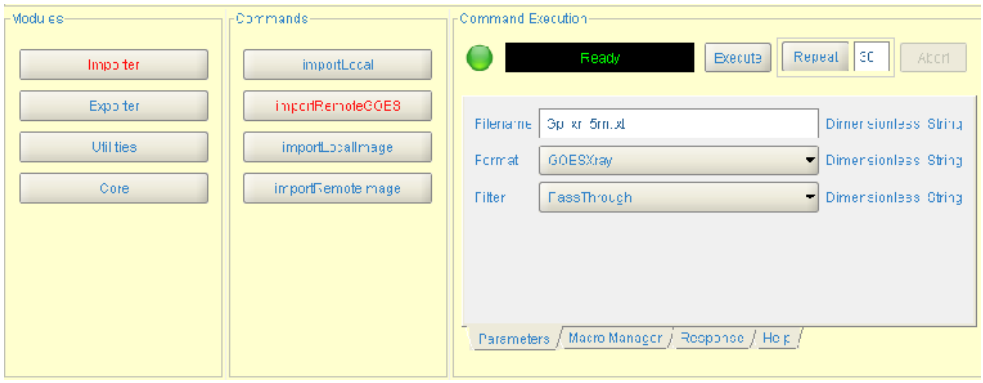
Here the nouns may be rocket engines or gyros, and the verbs may be life-changing, but the interface 'metaphor' is essentially the same as an HP oscilloscope or spectrum analyser. [36] [37] The user of the DSKY would enter the verb and noun, and perhaps some parameters for the command, say, '*fire the engine for three seconds*'. In Starbase this interface is achieved with descriptive words rather than numbers. So for instance if we had a set of commands for a `Builder` software module, the act of laying a brick could be written `Builder.layBrick`. Here the verb is conventionally in lower case, and the noun begins with an uppercase character. Suppose that the builder wanted to lay several bricks, the number could be passed as a parameter: `Builder.layBrick(3)` which lays three bricks. We have the beginnings of a programming language, as well as an easy to use interface for the instruments. This logic is carried through into the event logging, which records the target, the action, and the result (or error). These logs are easily parseable for later analysis.

The screenshot below shows the three main components of the UI. From the left, is the set of *Modules*. These are convenient groups of commands with a particular purpose. Next come the *Commands* within each selected module. Shown below, the Importer module is selected, and the commands available to the Importer. On the right, a set of tabs; the left tab allows entry of the *Parameters* for the command. The Response tab will show the data produced by executing the command.

Modules, Commands, Parameters

To complete the implementation of the design goal, the UI is described in a set of XML files. These

files specify the command set (i.e., the names, the parameters, data types, units and responses for each command, and so on). The experienced user can therefore change the appearance and behavior of an instrument to suit their needs or style of working. The detailed design of the command structure is beyond the scope of this paper, and probably would interest a developer more than a potential user. One important point is that new instruments can be built by carefully combining fragments of XML to allow access to commands in the 'command pool'.



You may be wondering something like, "Is that all?", because you have seen other software which has attractive interfaces for equipment with knobs, meters, sliders and so on. The intention is to make such higher level interfaces possible, once the underlying command structure is sorted out. I have to learn how to implement a scripting language to suit the objects in Starbase, and then macros can be added to visual components in the same way as for instance in Visual Basic. Only then can a more complex UI be created. The important point to remember from one of the Design Goals is that *all* aspects of the UI *must* be described in the XML files, so any macros, any widgets like knobs, *etc.*, must all start as an entry in the instrument description. There is no easy solution; the system has to be designed and built very much from the bottom up.

How do I make an Observation?

This section describes how to use the Starbase Observatory to make a simple observation (i.e., obtain some data from an instrument). These observations may be your own, in that you are responsible for the hardware and software for collecting the data, or it may have been made by another observer, for instance at a remote location.

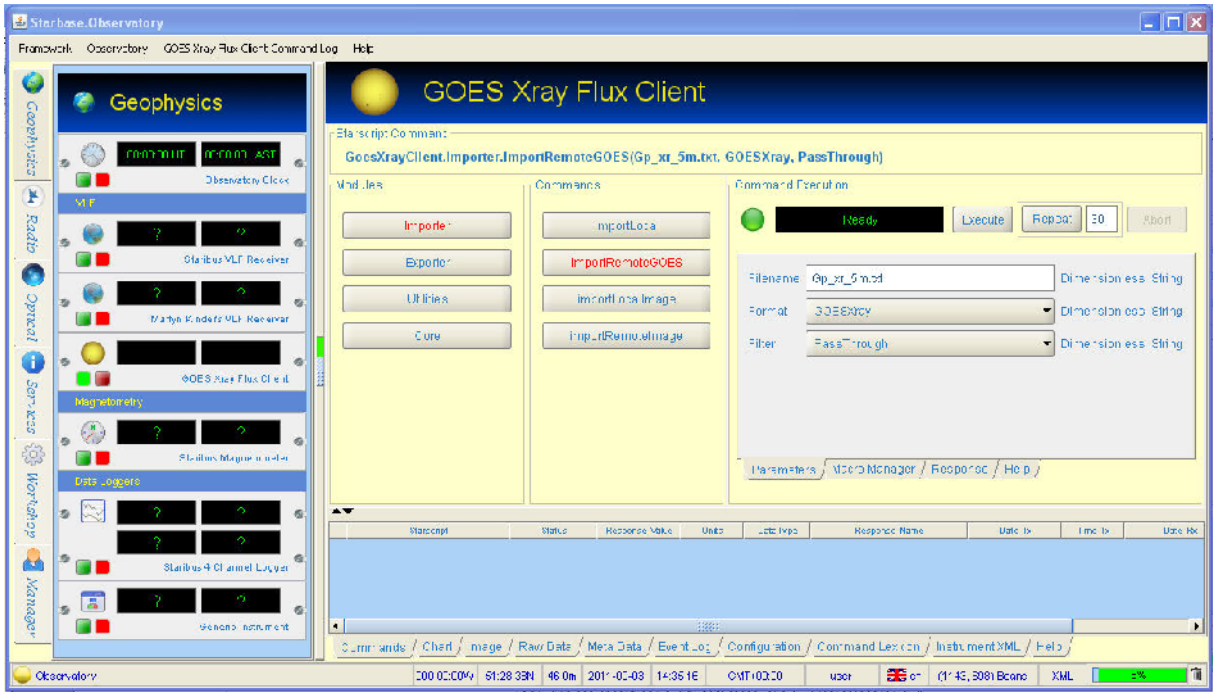
The first step in making an observation is to select an instrument to use. This example will use the GOES Xray Flux Client, a simple instrument which downloads data from the NOAA Space Weather Prediction Center FTP server [38]. This instrument is in the *Geophysics* group of instruments, on the top tab on the left of the screen. The Instrument 'control panel' appears as shown below:

The GOES Xray Flux Client Instrument

The Sun icon is simply a reminder of the purpose of the instrument, and is used on related screens. Note the start (green) and stop (red) buttons. The digital indicators show the *last* data values read.

First **select** this Instrument by clicking anywhere on this panel. The right hand side of the screen will change to show the Instrument's commands (screenshot below). **Start** the instrument by clicking the small green button in the lower left corner of the control panel (it will glow brighter). The instrument is now ready for use.

Starbase Instruments (with a very few exceptions) have a *Commands* tab as shown below, which is used to control the Instrument:



All Instruments respond to Commands

This example will show you how to download some Geostationary Operational Environmental Satellite (GOES) data from the NOAA Space Weather Prediction Center (SWPC) server [39].

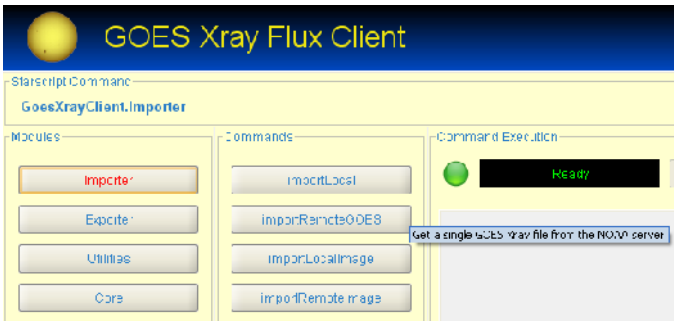
To set up and execute the command, firstly, select the `Importer` module from the left-hand set of command buttons. Notice that this instrument does not have a `DataCapture` module, since your observatory does not contain a satellite! (i.e., all data are obtained from files, either locally or via FTP).

Select the Importer module

This action will reveal those commands (in the center set) which may be used to read (i.e., `import`) data from several sources. In this case, we will read the data file from the NOAA server, so select the `importRemoteGOES` command:

Select the `importRemoteGOES` *Command*

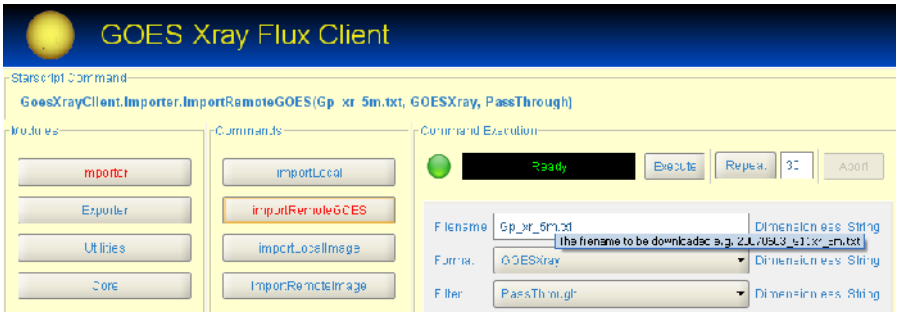
Notice how the command is being built up step by step, as shown in the window above the buttons. The command so far is:




```
GoesXrayClient.Importer.importRemoteGOES
```

This 'command language' is called *Starscript*, and will eventually be extended to allow composite commands to be executed as macros, thus simplifying multi-stage operations, and allowing users to exchange programs.

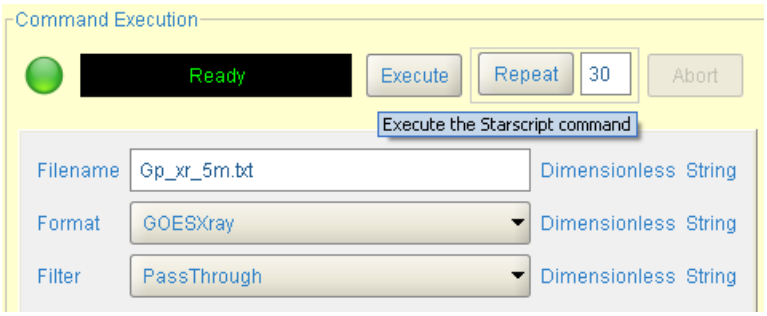
The next step is to specify the required dataset to download. The instrument Help tab will give further information on the options here, such as filenames. The easiest option is to take the data from the primary (or default) satellite, as shown below.



Specify the dataset to download as the `Filename` *parameter*

There is only one data format available here, so the `Format` parameter must be `GOESXray`. The `Filter` will either take the data unmodified, or filter with a `SimpleIntegrator`. In this application, it is best to use the `PassThrough` filter, which will not process the data in any way.

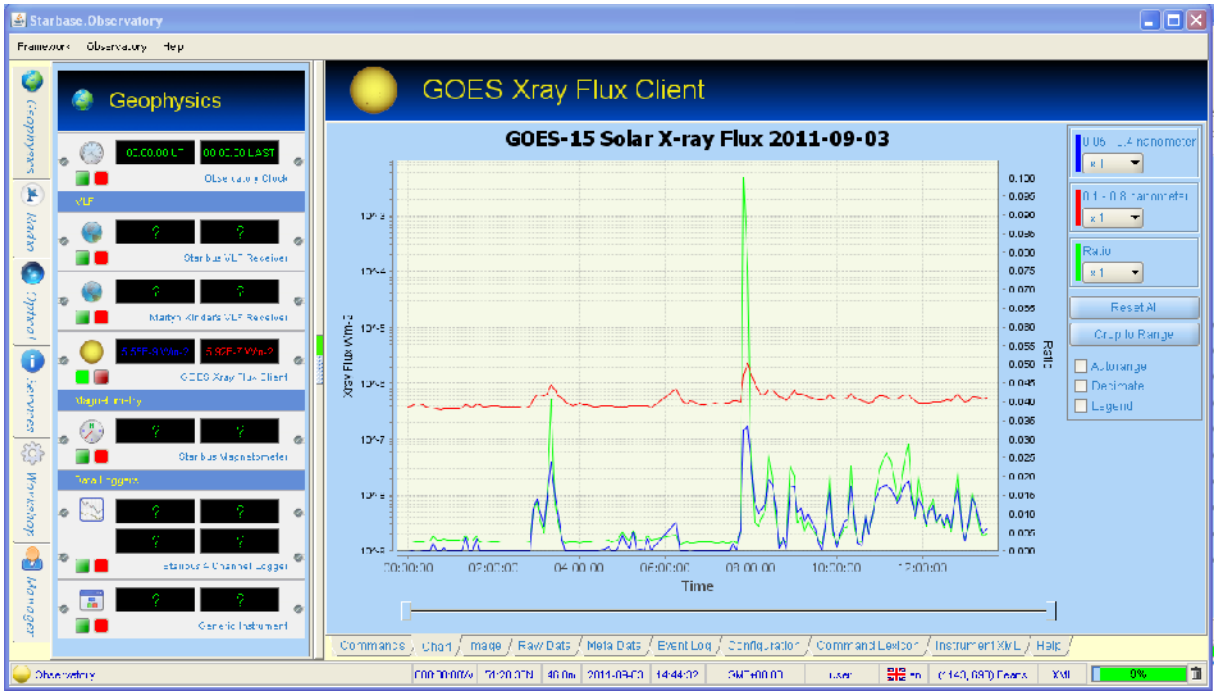
The full *Starscript* text of the command is therefore:




```
GoesXrayClient.Importer.importRemoteGOES
(Gp_xr_5m.txt, GOESXray, PassThrough)
```

Execute the Command

Finally, execute the command by clicking the **Execute** button. Provided that the server is available, you should see a `SUCCESS` Status near the right hand side of the Command Log, and the data will be available on the Chart and data tabs. If all of the previous steps have been followed correctly, then you should see the current version of the chart below, giving the X-ray flux from the Sun at two wavelengths, and the ratio of the fluxes in each channel. The *RawData* tab will tabulate the data samples in the file. Notice that the chart title and other annotations are taken automatically from the metadata accompanying the data.



Solar X-ray Flux from GOES 15

If you are making solar observations using other instruments, then these charts (and the associated data) are a very useful resource against which to compare your results. It can be very rewarding (and reassuring) to find that you have captured an event which was also seen by the satellite.

Where is it Going?

You can see the short-term roadmap for the project by visiting the Mantis site mentioned earlier. Some of the highlights are below:

- Addition of calibration parameters to the Metadata dictionary.
- Accurate mapping (and inverse mapping) between FITS and RSP metadata and the Starbase Metadata dictionary.
- Good support for FITS images and data are seen as essential, but complex to implement.
- The FUNcube dongle would be another useful data source, but is challenging because of its USB interface, which is difficult to implement cross-platform.
- The addition of a VLF signal generator and spectrum analyser (via a sound card) would be of great help to the UKRAA VLF Receiver users.
- Improved context-sensitive Help information (currently very sparse).

The essential longer-term 'missing links' (in no particular order) are likely to be:

- Macro facility for full scripting automation.
- Central data Repository (Starbase becomes a database client).
- Data analysis tools (macros would help with this).

If you would like to be involved in the Starbase development, then please contact:

`starbase@ukraa.com`

More Java developers would be very useful; experience with XSD, XML, XMLBeans, SVN, Git or MySQL would be a real bonus. Cross-target development in C is also required for hardware plug-ins. You are also welcome to donate to the project via the UKRAA website.

Who made It?

Starbase has been evolving for at least ten years, and many people have contributed ideas, criticism, suggestions and moral support. I would particularly like to thank David Farn (BAA RAG) for his work on getting the *Staribus* protocol defined and working successfully. Thanks also to my colleague, friend, and radio mentor Alan Melia (G3NYK) who has been a real support for several years, a sounding board for my silliest schemes, and expert at breaking that

which I believed to be unbreakable; we have also built some challenging pieces of hardware. A recent recruit, Mark Horn, our cross-platform and server specialist, has ensured we work in a more professional manner, introducing new ways of working and tools, and much-needed raw enthusiasm. My previous manager at BT, Tony Baines, a Java professional and Agile coach, persuaded me to move the entire architecture to be schema-based. At the time I was overwhelmed by the work required, but I am very grateful for his insight, suggestions and support during the transition. Many RAG members have tested and tested again, in particular Andrew Lutley and Martyn Kinder, whose feedback stimulated yet more improvements. And finally, the Open Source community whose libraries I have freely plundered to my own ends, saving me hundreds of hours of time.

I would also like to take this opportunity to thank the Council of the BAA for awarding me the Horace Dall Medal (right) in 2009, "*... to a person who has shown marked ability in the making of astronomical instruments*", which was a gratefully-received vote of confidence in the project. [40] [41]

Where can I find more Information?

[1] http://www.ukraa.com/www The UK Radio Astronomy Association
[2] http://www.britastro.org/radio/index.html The BAA Radio Astronomy Group
[3] http://www.britastro.org The British Astronomical Association
[4] http://www.funcubedongle.com The FUNcube dongle receiver
[5] http://www.w3schools.com/schema An introduction to XML and XSD schema
[6] http://atadastral.co.uk BT Research Centre
[7] http://www.forth.org The now defunct FORTH Interest Group
[8] http://www.britastro.org/radio/downloads.html RAG Circulars describing the *Plug and Play Observatory*
[9] http://www.modbus.org The Modbus Organization
[10] http://jamod.sourceforge.net/kbase/protocol.html The Modbus protocol
[11] http://www.ukraa.com/www/downloads.html Staribus Draft Specification
[12] http://www.endurancegb.co.uk Endurance GB, UK's Endurance Racing
[13] http://www.sportident.co.uk SPORTident suppliers of event timing equipment
[14] http://www.gnu.org/copyleft/gpl.html GNU General Public Licence
[15] http://www.apache.org Apache Software Foundation
[16] http://en.wikipedia.org/wiki/Self-similarity Self similarity and fractals
[17] http://www.iso.org/iso/date_and_time_format ISO 8601 Date and Time formats
[18] http://www.npl.co.uk/reference/measurement-units Système International d'Unités
[19] http://articles.cnn.com/1999-09-30/tech/9909_30_mars.metric.02_1_climate-orbiter-spacecraft-team-metric-system?_s=PM:TECH Loss of Mars Orbiter
[20] http://en.wikipedia.org/wiki/Open_source The Open Source initiative
[21] http://en.wikipedia.org/wiki/Source_lines_of_code Lines of Code estimation
[22] http://subversion.apache.org Subversion version control system
[23] http://www.ukraa.com/www/starbase/websvn.html Web SVN of Starbase source code
[24] http://www.ukraa.com/bt/view_all_bug_page.php MantisBT view of Starbase issues
[25] http://en.wikipedia.org/wiki/Agile_software_development Agile Programming
[26] http://en.wikipedia.org/wiki/Metadata Discussion about metadata
[27] http://fits.gsfc.nasa.gov FITS Support Office
[28] http://www.radiosky.com RadioSky Pipe
[29] http://heasarc.gsfc.nasa.gov/docs/fcg/standard_dict.html FITS dictionaries
[30] http://heasarc.nasa.gov/docs/heasarc/ofwg/docs/general/ogip_93_001/ogip_93_001.html Specification of Physical Units within OGIP FITS files
[31] http://ccat.sas.upenn.edu/~haroldfs/messeas/implement/communicationfailure.htm Chinese Whispers
[32] http://www.radio-astronomy.org/node/203 SARA RASDR Software Defined Radio
[33] http://www.moetronix.com/spectravue.htm SpectraVue software
[34] http://www.qsl.net/dl4yhf/spectra1.html Spectrum Lab software
[35] http://www.usabilityfirst.com/glossary/noun-verb-paradigm/ Noun Verb User Interface
[36] http://en.wikipedia.org/wiki/Apollo_Guidance_Computer Apollo AGC
[37] http://www.ibiblio.org/apollo Apollo AGC
[38] http://www.swpc.noaa.gov/today.html Space Weather
[39] ftp://ftp.swpc.noaa.gov/pub/lists/xray GOES Xray FTP server

[40] http://adsabs.harvard.edu/full/1987JBAA...97...76H Horace Dall
[41] http://www.britastro.org/exhibition/report2009/index.htm Horace Dall medal presentation

Glossary

BAA	British Astronomical Association
CSV	Comma Separated Value
DSKY	Display/Keyboard
FITS	Flexible Image Transport System
FTP	File Transfer Protocol
GMT	Greenwich Mean Time
GOES	Geostationary Operational Environmental Satellite
GPS	Global Positioning System
HP	Hewlett Packard
HTTP	Hypertext Transfer Protocol
JAR	Java Archive (a special kind of ZIP file)
NMEA	National Marine Electronics Association
NOAA	National Oceanic and Atmospheric Administration
NTP	Network Time Protocol
RAG	Radio Astronomy Group (of the BAA)
RS232	Electronics Industry Association Communications Protocol Standard (currently, EIA-232)
RS485	Electronics Industry Association Communications Protocol Standard, differential transmission (currently, EIA-485)
RSP	Radio Sky Pipe
RSS	Really Simple Syndication
SDR	Software Defined Radio
SI	Système International d'Unités
SQL	Structured Query Language
SWPC	Space Weather Prediction Center
TSV	Tab Separated Value
UKRAA	The UK Radio Astronomy Association
UI	User Interface
USB	Universal Serial Bus
VLF	Very Low Frequency
XML	Extensible Markup Language

About the Author: Laurence Newell lives in Suffolk, England. He trained as a physicist, with a research project involving high-power lasers and plasma diagnostic techniques. His career with British Telecommunications has covered diverse areas such as the Global Challenge yacht race management software, testing Ethernet-based communications hardware, security reporting systems, the design of a Fast Walsh Transform processor, the design, development and marketing of microprocessor control hardware and software, human factors aspects of UI design, and even some pages of bt.com. He has built several radio telescope systems, from a 151 MHz phase-switched interferometer, to a 1420 MHz Hydrogen Line receiver, and would like to build an autocorrelation spectrometer. His main interests are in data processing, and the associated telescope control systems. He was also involved with the reformation of the BAA RAG and the creation of UKRAA, and is now concentrating on the development of Starbase. If you would like to contact me with any questions, please email me at starbase@ukraa.com.