University of Edinburgh
Division of Informatics

Improved data logging, sharing and analysis for the British Geological Survey's School Seismology project

4th Year Project Report
Software Engineering

Jon Gilbert

March 11, 2011

Abstract: This report presents a replacement data logging and sharing system for Schools Seismology Project run by the British Geological Survey. The currently supplied software (AmaSeis), requires a dedicated PC running full time within each school, while the devised solution runs instead of a small device consuming a much smaller amount of power. The solution also attempts to solve some of the problems with the current functionality to automatically send data to a server at the BGS for analysis, in a way more likely to be permitted by restrictive school firewalls and also aims for portablity. The AmaSeis client familiar to the teaching staff can also still be used, connecting to a replacement server on the device.
The requirements of the project are provided and detailed information on the investigation, design, implementation, delivery and evaluation of the produced system.

Acknowledgements

Firstly, I would like to thank my supervisor for this project, Dr Michelle Galea, for her advice and support during the year and for assisting in solving the significant number of technical problems that arose. Further to computing support staff were always efficient in helping to solve these issues.
Secondly, I would like to thank Dr Paul Denton and Dr Aoife O'Mongain from the BGS for meeting with me to discuss requirements and review progress and for taking the time to help test the project.
Finally, I would like to the author of AmaSeis, Dr Alan Jones for providing access to the source code of his work.

Contents

1. Introduction

1.1 Summary

This report presents details of the completed project to replace the data logging and sharing components of AmaSeis[22]. In this chapter a brief introduction to the School Seismology Project is provided along with an analysis of the problems that exist with the current software.
Section 2 explains the requirements of the project, starting with the original specification, then explaining the changes that were made and listing the final requirements. Details of how the requirements have evolved over time, including discussions with the British Geological Survey (BGS)[1], the recipients of the system, are presented.
In section 3 details of the work carried out to investigate potential solutions are provided. Comparisons of the available choices and explanations of final decisions can be found here.
Section 4 explains the process of designing the system from initial prototyping, refinement and presents an overall stucture of the system. Finer detail on the technical aspects of the code is provided in section 5, which also provides information on tools used during the development and testing of the solution.
Section 6 provides an evaluation of the completed system, how well the requirements were met and what limitations exist. Details of a follow up meeting help to evaluate the system are also provided. Section 7 then explains the work undertaken to package the system for delivery, allowing for duplication and the setup of new hardware units.
Finally section 8 explores the potential uses of the data for the BGS and other work being carried out to update AmaSeis.

1.2 School Seismology Project

The BGS operates a scheme known as the School Seismology Project[39], which allows schools to collect data from locally sited seismometers, capable of detecting signals from large earthquakes occurring anywhere in the world. This data can then be analysed by the provided software and used as an aid to teach both geography and physics concepts to pupils. Data can also be shared with other schools around the world via a central server which stores collected files.
The project has been running for a number of years and there are currently over 400 schools in the UK participating.

1.3 AmaSeis

The software provided for use with the seismometer is known as AmaSeis[22], which interfaces with the device via a serial port (an analogue to digital converter is used between the seismometer and the serial port) and records incoming data to a custom file format known as an hour record. AmaSeis provides a GUI which allows users to browse data in a view similar in appearance to that of a drum or heli recorder, perhaps familiar as needles plotting ink lines on paper. AmaSeis provides a simple way to navigate through the available data via scrolling and zooming and has GUI controls to allow users to jump to a certain date and time. AmaSeis shares data via the File Transfer Protocol (FTP) to a central server provided by IRIS[2]. The software automatically uploads a file at the end of each hour (i.e as it is completed) and can retrieve shared files from other participating schools via the same protocol.
AmaSeis also implements a client/server model to share data locally within each school. The machine performing data logging is setup to run in server mode, with clients connecting over the local network to retrieve data files. The same software is used for both purposes, with a few settings to be changed within the program to determine its mode.
A number of seismometers are supported by AmaSeis, the most popular one is use in UK schools being the SEP[25] device. AmaSeis is used extensively in other countries, most predominately as part of the IRIS Seismographs in Schools project[21] in the United States, which typically employs a different model of seismometer.

1.4 Current Problems

AmaSeis suffers for a number of limitations. Firstly the software runs only on the Windows operating system, limiting the options for deploying it on a device other than a PC. As schools are required to run the software on a PC and data logging is a continuous activity, the cost in electricity is estimated to be around £200 per annum. With saving energy now a concern for many organisations, both in terms of cost and environmental impact, an alternative that uses a minimal amount of power while freeing up a PC for other uses is highly desirable.
The method of data sharing over FTP is often hampered by school firewalls, which can be very restrictive, leaving users with access only to their own local data. Data sharing only attempts to send the hour which had just been completed and must be performed manually if older files are to be sent to the server, for example if the system has not been set to upload previously or if it was otherwise unsuccessful in attempting to transmit the files.
The default format for storing data used by AmaSeis is also restrictive. The

files are in a custom format, only of use to other AmaSeis clients, rather than using one of the standard formats. The client is however capable of reading files from formats which are more commonly used in seismology such as SAC[19] and others. The software can also export data into these formats, albeit manually, via a 'Save As' dialog, though there is no option to record data directly in this way in the first instance.

1.5 Further Information

The setup and operation of the SEP seismometer and AmaSeis, as well as basic information on how to analyse the recorded data, is well documented. The manual is available online from Middlesex University Teaching Resources[25], who are the suppliers for the system in the UK.

2. Requirements

2.1 Original Specification

The original project specification[41] was broad in scope. It aimed to achieve the following:

- Perform data logging a low key manner, such that it can run on small, cheap hardware that consumes little power.
- Automatically share data via a central server, hosted by the BGS.
- Only share data only when an earthquake could have been recored.
- Use HTTP as the protocol over which to transmit data, as most firewalls will permit this.
- Provive a platform on which to perform analysis of the data via a web portal, using a system such as Rapid[6], so that school based users can work with data via a web browser.

This set of goals sought to completely replace AmaSeis with a web based solution, with all analysis being performed on the remote server and only data logging and transmission being performed locally in schools.

2.2 Revised Specification

2.2.1 Meeting

In late October 2010 a meeting was held at the Edinburgh offices of the BGS in King's Buildings. The purpose of this meeting was to report on progress to date, clarify the project requirements and to aid in my understanding of the project generally, including the problems with the current system. It was also an opportunity to discuss ideas not outlined in the original specification and devise a rough time line for the project.

2.2.2 Outcomes

2.2.2.1 Alterations

As a result of the meeting some significant changes were made to the project requirements. Firstly the aim of producing an operational analysis portal was thought to be overly ambitious, given the time constraints of the academic year. As such this requirement was dropped with the provision of such a site to be left as task for later work. The primary aim, as mentioned in the original requirements, remained the data logging and sharing components.
As an extension to the project, time permitting and in place of the web portal, it was put forward that allowing AmaSeis clients to connect to the new system directly would still achieve the aim of freeing a PC from use as a server, while allowing teachers access to local data, using software with which they were already familiar. Before committing to this requirement some investigation was required to determine how feasible such functionality was, as the details of how this sharing operated were unknown at this time. To that end Alan Jones, the author of AmaSeis, kindly provided the source for his work soon after this meeting.

2.2.2.2 Data Format

The desired file format to be used for data logging was determined at the meeting. The BGS requirement was for the miniseed[18] format, part of the SEED[20] international standard for seismological data. Each file contains meta data in the header, providing the station name, number of samples, start and end times and network, as well as other information, allowing files to independently describe the data they contain. This is in contrast to AmaSeis hour records, which simply contain four bytes for the number of samples within followed by the data itself. AmaSeis relies on the directory structure and filename to determine the origins of the data, for example 2011/01/03/16.Z is 4pm on 03/01/2011, with a component of Z. Miniseed files have an additional advantage in that they can also be of any length, allowing shorter or longer time spans to be covered by each file. Data compression within the files is also supported, using a method known as STEIM2 encoding, to record only the difference between values, representable in fewer bits than a series of whole integer values.
There are a variety of other data formats used for storing seismological data, SAC[19] in particular was mentioned but it was felt miniseed would be the easiest to deal with and that tools for converting between formats, if this is required later, are readily available and can be used on the data once collected.

2.2.2.3 Licensing

Software licenses were raised and decided upon at this meeting. It was agreed an open source license would be ideal, so that modifications could be made later on without any copyright issues arising. The General Public License version 3 was chosen (GPLv3)[11] for any software resulting from the project. As some of the already available libraries which appeared potentially useful were released under the same license, this choice would also help ensure straightforward compatibility. These libraries are discussed in section 3.2,

2.2.2.4 Configuration

The configuration of the devices should be relatively straight forward for end users. A web interface, similar to that which might be found on a home network router was a requirement, as the device, not being a PC, would be unlikely to have its own display or input devices. It would also be useful if the end user were able to download recorded data locally via this interface.

2.2.2.5 Other Considerations

A few other points were raised at the meeting which can be specified as requirements. The first of these is that data logging requires accurate time keeping. Most modern operating systems make use of the Network Time Protocol (NTP) for synchronising clocks. Older versions of Windows, the platform on which AmaSeis is operated, typically lacked this feature and additional software was needed to perform this task.
It was also noted that there are a number of different seismometer models available and in use but that the only one available for development of the project was the SEP device, the model used in the School Seismology Project. It may however be desirable at a later date to interface with another seismometer and design of the software should take this into account.
A receiving server is also required to accept files from clients based in schools and store the data for the BGS. The only functionality required here is the ability to authenticate and store files. Processing of the data will be left to the BGS.

2.3 Finalised Requirements

The following represents a more concrete and final list of requirements which have evolved over the course of the project, mostly from the original specification and the October meeting but also refined via email conversations with BGS

staff. Some of these requirements are based on information discovered during the investigation phase of the project (see section 3).

2.3.1 Hardware and System

- Implement the system on a hardware device which is cheap to buy and consumes little power.
- Ensure system is capable of keeping its clock synchronised (most likely via NTP).
- All times recorded by the system should be as Coordinated Universal Time (UTC).[42]
- The produces system should aim for portability, should the chosen hardware become obsolete or unavailable.
- System startup is automated, once the hardware is powered up and seismometer connected, data logging should commence.

2.3.2 Data Logging

- Interface with and log data from a SEP seismometer via a serial port.
- Record data in the miniseed format.
- Send data, via HTTP, to a server process running on a BGS computer.
- If unable to currently send for any reason, keep the most recent 12 hours worth of data available for when it is possible to transmit again.
- Once sent, archive data for as long as is possible.
- Automatically remove the oldest data when storage space is limited.
- Provide a web interface allowing for the following
  - Access to local data archives.
  - Configuration of the software.
  - System log viewer.
  - Time synchronisation information.

2.3.3 Receiving Server

- Provide an HTTP based server to listen for clients.
- Authenticate requests to upload files.
- Accept incoming miniseed files and store them to disk.
- Files should be stored by year and day, optionally also by station.
- Provide a means of managing the authorised client list.

2.3.4 AmaSeis Server

- Log data in the AmaSeis hour record format.
- Allow existing AmaSeis clients to connect to the device using the current protocol.
- As with miniseed files, remove old data to allow space for new files. The limit for AmaSeis files should be a maximum age of one year.

3. Investigation

3.1 Hardware

3.1.1 Requirements

In investigating possible hardware the goals of the project had to be taken into consideration, that is the solution must run on a cheap and low powered device to save on electricity and free up a PC and be affordable (justifiable within a school budget) for end users. Beyond these attributes in order to interface with the seismometer a serial port was required, with the use of a serial to USB converter a possibility, as old style serial ports are becoming rare on newer devices. The hardware should also be readily available and likely to continue to remain on the market for the foreseeable future.
Storage space is also an issue. The SEP seismometer produces samples as 16 bit integers at a rate of 20Hz. This equates to around 3.4Mb per day in raw data, with other seismometer models producing data of similar magnitude. Other considerations are the compression offered by the miniseed format and the space required for file headers (meta data). If 4Mb per day is assumed as a conservative estimate, then one years worth of data should consume approximately 1.5GB of storage space. Once files are logged and transmitted it also possible that they can be grouped together and compressed further in archive files, reducing this requirement further.
It should be noted that once data is transmitted to the BGS that the storage of it on the device is for the use of end users only. Since the data exists on the server, the deletion of old data to save space is acceptable, though a device which could allow for extended storage for users who wish to keep local data would be advantageous.

3.1.2 NSLU2

The suggested hardware solution (from the project specification) was the NSLU2[45], launched by Linksys in 2004. It is a network attached storage (NAS) device, with an embedded 266MHz CPU, 32Mb of RAM and 8Mb flash storage, network and USB interfaces, and is primarily designed to be attached to external hard disks, allowing access to them over a network. The device gathered around it a significant community of developers who modified it for a variety of purposes, including mail and web servers, weather stations and firewalls. By default the device runs a

customised version of Linux and it is possible to replace this with other distributions. Typical power consumption of the NSLU2 is less than 10 Watts, although as storage is via external drives, these also require power. The NSLU2 unfortunately was discontinued in 2008 and only second hand models can be found still on sale.

3.1.3 Other NAS devices

Given the discontinued status of the NSLU2 the next step was to investigate other and more recent NAS devices which could replace it. A popular alternative, still in production, was the Buffalo Linkstation[10], of which there are several models. The Linkstation provides a more in terms of CPU power, memory and internal storage but at the cost of some additional power, stated by the manufacturer at between 15 and 45 Watts (depending on model) and without any USB devices attached.
The cost of a new device, (cheapest available model being the 'Live LS-CHL 500GB', is around £75. These devices seem to have attracted a community of developers in a similar way to the NSLU2, with various modifications being made to them by a number of users. A significant benefit over the NSLU2 is that the disk drive is internal (with extra drives attachable via USB) and this disk has more than enough space given the data requirements. It appears likely that the manufacturer will continue to sell these devices for some time into the future. Exact specifications on the internal system resources, aside from the disk storage space, do not appear to be readily available from the manufacturer.

3.1.4 Sheevaplug

Further exploration of the hardware currently available led me to the Sheevaplug[40], marketed as a 'plug computer' and designed to provide a small, low powered (stated at 5 Watts) Linux based system. The standard Sheevaplug provides a 1.2GHz CPU, 512Mb RAM, 512 Mb flash storage and the same USB and network interfaces as would be expected on a NAS device.
The difference between the Sheevaplug and NAS hardware is that it has been designed as a general purpose device, more easily configurable for a custom task. There is additional hardware to support an SD card interface and a mini USB port, allowing developers to connect to the device over a serial connection.
The cost of a standard Sheevaplug is around £95, with extra costs for the SD card and again it has attracted a reasonable sized community supporting development on it. The SD card support allows for reasonably sized and extendible storage for data, as well as of easily duplication and distribution of any software. The device can be set to boot directly from a card, or from a USB device as desired.

Given the advantages in configurability the Sheevaplug was chosen as the platform on which to base this project. It also consumes the least power and is the smallest of the devices described. The disadvantages are a slight increase in initial cost and the reduced storage capacity versus the Linkstation.
There are alternative but similar devices available from the manufacturer, such as the Guruplug[40], which provide higher specifcation, through additional USB, network and SATA ports, but at a higher cost. For this project the original Sheevaplug provides more than enough resources and so anything further is unnecessary at this point. Although these devices originate in the United States, a supplier exists in the UK, named NewIT[3].

3.1.5 Cost vs PC

The estimated cost of running a standard PC 24 hours a day to perform the required data logging was placed at around £(200). The exact power consumption and cost is dependent on the configuration of the PC in question and the cost of electricity, both of which will vary. At the time of writing the cost per kilowatt-hour from one of the large suppliers was 12.7p, so this estimate points towards a PC consuming around 180 Watts, broken down as follows. Here 8760 is the number of hours in a year and 12.7p is the sample price mentioned previously.

$$180 \times 8,760/1000 = 1576.8kWh$$

$$12.7p \times 1576.8 = £200.25$$

For the Sheevaplug at 5 Watts, the cost is £5.56 derived as follows.

$$5 \times 8,760/1000 = 43.8kWh$$

$$12.7p \times 43.8 = £5.56$$

The calculations are simplified, costs will vary over time and pricing models are more complex than a single price per unit. However the size of the savings will certainly cover more than the cost of the device within the first year of operation.

3.2 Software

Investigations into work already available in other libraries proved useful. IRIS provide a significant number of toolkits and APIs, along with detailed manuals[17] for various purposes, including the access to data in the various standard file formats used. Much of the available tools centre on Java as the language of choice,

with others in C or Matlab.
The Obspy[5] project provides a python[32] framework for seismological applications. In particular is provides support for manipulating data in a variety of file formats, including the required miniseed format. Also provide are modules for imaging, that is plotting data and exporting it to image files and database/web clients for connecting to various internet based services.
Libraries were also looked at for interfacing with the seismometer via a USB port. Given the prevalence of USB devices and standardisation of these ports it was difficult to find a language in which there was not some form of support for this task. The task of dealing with HTTP requests, required for sharing data, is also a standard feature in most languages.

3.3 Programming Languages

The default installation of the Sheevaplug provides a reasonable choice of programming languages. Some are provided as standard, while many others are available to be installed. Some such as Java, rely on community efforts to port them to the device, with open source runtime environments and development kits available. For some languages a cross-compiler is provided, allowing compilation to be performed on a more powerful desktop machine, before being transmitted to the device for execution.
The final choice of language was python. It was selected for the breadth of available libraries, in particular the Obspy project and for my own familiarity with the language. This choice also supports the requirement for portability, with python available for a wide variety of systems and the lack of any need for software to be compiled allowing for simple transferal of the system to a new platform, assuming python can be installed on it. Further, given the choice of available python versions it was decided to produce code which would run on version 2.6, while also making it compatible with the most recent version 3 release.
An attempt was made throughout the project to follow the PEP 8 style guide[36] for writing python code. The aim being to produce readable and clean work should anyone else wish to modify the project at a later date.

3.4 Operating System

As supplied the Sheevaplug runs a somewhat cut down version of the Ubuntu Linux distribution[8] compiled for the device. Other distributions are also available as community efforts, with Debian appearing to be the most popular of these. As Ubuntu provided everything that was required for the project, it was

decided that no changes were needed here, except to upgrade the kernel from the supplied version, to a more recent one (from 2.6.30.2 to 2.6.37.1). Details on this process and general configuration is discussed in 7.3

3.5 AmaSeis Server

As a result of the initial meeting with the BGS, the feasibility of implementing the AmaSeis server mode in the new system was to be investigated. There were two requirements discovered to make this possible. The first was the storage of data into the AmaSeis hour record format. Doing so would require additional storage space, approximately double the original amount. The second was implemented a TCP based server which could deal with the request from clients.
The protocol itself is quite simple but fragile. A handshake take place in which the client initiates contact and the server provides the information on the current time, station name and location, sample rate etc. The client then begins to request files for each hour in the current GUI view and the server provides either the file size and data or a zero file size value to indicate it does not exist. The client also regularly makes calls to fetch the current data, that is data recorded in the current hour, for the file which is not yet complete. This is achieved by sending the current number of samples for the hour that the client currently has, the server comparing this versus its own data and the difference, if any, being returned to the client to append to the current record it has.
The operation of the protocol is prone to problems but appears to work well enough for use over a local network. There is no real error checking and no recovery from errors, for example if the client requests a file and the server does not respond correctly or at all, the client program will tend to hang, where perhaps a timeout and error message would be appropriate. As it is only the server that was to be reimplemented, attempts to fix these issues were not possible.

3.6 Receiving Server

3.6.1 Purpose

Separate software was required to receive data from the school based devices, capable of accepting files over HTTP. As a service available publically on the internet, security was an issue and a protocol to handle authentication and response to clients was needed (see section 4.3).

3.6.2 Operation

Receiving data requires a web server which can accept accept incoming requests, authenticate the client and store the miniseed data to disk. At the time of the October meeting the nature of the server on which the software would be deployed was unknown. Given this lack of information on the environment in which it would be used, two separate solutions were devised. The first of these was written in PHP[28] and consisted of a collection of scripts to be used on a standard web server setup, such as Apache, which may be shared with other BGS websites. The other, written in python was a fully independent web server, should the target machine not require to be running a more heavyweight solution. Both implementations provide the same functionality and the same protocol for client/server interaction .

3.6.3 Data Storage

The transmission of data from a number of clients raised the question of how much storage would be required on the receiving server. Each client transmits a 4kb miniseed covering a period of around 90 seconds, or 40 per hour. One client will transmit 3.75Mb of data per day, or 1.36Gb per year. As the existing system is in operation in over 400 UK schools the potential storage requirements are large, if the BGS wish to keep the data permanently.
File organisation is also an issue here, since a client sends around 960 files per day. The receiving server implementation allows for storage of incoming files by date only (/year/day/file.mseed, where day is between 001 and 366) or date and station (/year/day/station/file.mseed). Filenames also represent the station name and start time of the file, even though this is described within the headers. This is done so that that files do not need to be opened to determine their contents, would could be slow over such large numbers of them.

4. Design

4.1 Prototyping

In the early stages of the project, in order to aid my understanding of the operation of the seismometer and the requirements, a prototype data logging system was created. This functioned only to connect to the seismometer and write the results to file as text, with a timestamp-value pair on each line. Each file covered one hour (in much the same was as AmaSeis) and the use of storage space was not particularly efficient. While this system was limited and flawed it provided a useful basis for early discussions and a platform on which to build the software.

4.2 Structure

The software for data logging, transmission, and the servers for the web interface and AmaSeis were divided into seperate processes. This was down to simplify the implementation of each component and because, for the most part, the is little need to share data between each seperate task, with the exception of the completed data files from data logging to transmission and AmaSeis server. File level locks are employed to prevent race conditions, for example if a data file is only partially written and the sender attempts to read it. Another option would have been to use threads for these tasks, but the potential for problematic debugging and the small CPU time required by the tasks suggested the extra overhead of using individual processes would not be an issue.

4.3 Data Transmission Protocol

The task of transmitting logged data to a central server required an application layer protocol to be devised. An overview of the operation of this protocol is provided in figure 4.1 on page 18.
The authentication relies on a shared secret between the client and server. New stations must be added to the authorised list, with a station ID and password combination. On sending a file clients sign the data with a hash generated (via the md5 algoritm[43]) from the combined hashes of the password and file. Each combined hash is unique and only valid for the file being sent and the password cannot easily be determined. The receiving server can then compare these values to both authenticate the station and determine the integrity of the file. Although

md5 has been shown to be susceptible to collision attacks, the knowledge and effort required defeat this form of protection is substantial, while the likelihood of someone attempting it is small, as there are no gains to be made from attacking the system. This method of signing the data also allows it to be sent in plain text, removing the need for encryption to be used on the channel.
It would have been possible to secure this protocol more fully by implementing a complete digital signature scheme or by enforcing SSL (HTTPS) connections between client and server. Given the minimal risks however and the added costs of such efforts the provided protocol was felt to be sufficient.

Client:
1. Send: station-id, filename and file-data, h(h(password)+h(file)) as combined-hash and h(file-data) as file-hash

Server:
1. Lookup: Hashed password for station-id as matching-password
2. Compare: h(file-data) versus file-hash
3. Compare: h(file-hash + h(matching-password)) versus combined-hash
4. Respond with: OK or FAIL
5. Store file if OK

Client:
1. Archive file on OK or retry later on FAIL or timeout

Figure 4.1: File transmission protocol

4.4 Web Interface

4.4.1 Configuration

End-users need to be able to configure their device, for example setting the station ID and password. A web server running on the device was required for this purpose, which users could access via a browser over the local network. Users are presented with a form in which they can alter the current configuration values and input validation seeks to prevent bad inputs being saved, with meaningful errors provided in such a case.

4.4.2 Device Status

Another required feature for the interface was the ability to view the current status of the system. While logging and transmitting data the software writes entries to a log file, which is made available to end uses via a page in the interface.

This page also shows the current status of the time synchronisation software. When operated on a platform on which this is not available, the user is unable see this information and is informed instead that it is unavailable.

4.4.3 Access to Miniseed Files

Allowing end users access to the recorded data locally was another requirement solvable via a web interface. After miniseed files are created and sent by the system they are compressed and archived in zip files, spanning one hour each. Archives files are created by the sender process, which performs the operation after a number of files are ready to be stored, this allows for more recent data to be accessed by the user, although the file download may be updated afterwards. Zip was chosen as format here as it is the most common likely to be familiar to Windows users (which schools typically use) and is available by default on newer versions of the operating system.
Access to files is by date and time, with users presented with a list of available years, then months, days and files per hour as they select each item respectively.

4.4.4 Help

Basic instructions are provided to users on the homepage, explaining the various functions of the interface and directing them to the appropriate pages to perform the required tasks. The provision of more detailed documentationm, if it is desired, is a task left for the BGS.

4.4.5 Security

Authentication is required for access to the interface. As access to the device should only ever be over a local network, within a particular school, the risk of malicious acts is low. As such only `Basic HTTP Authentication`[13] is employed, which passes the authentication token in plain text (encoded as base64). This has the advantage of avoiding any saving on state for user sessions with the web server operating on the device and dealing with the overhead of encryption.
There is only one accepted user, with the username is set to `admin` and a password which can be altered from within the configuration management page. Allowing for multiple users was considered but did not seem necessary for this project.

4.5 Portability

One of the non-functional requirements specified for the software was that it should be as portable as possible. Using python as the development language helped to achieve this. In designing the software however there are a number of differences, particularly between Windows and Linux which needed to be addressed.

5. Implementation

5.1 System

5.1.1 Time Synchronisation

Completing the task required more than just creating the data logging software, other considerations, such as ensuring the system maintained the correct time also needed attention. By default the Sheevaplug, or at least the version received at the time the project was started, did not make use of NTP[44] and installing and configuring it (adding additional servers) was required.
At one point in the project a significant problem developed in which the system clock advanced at a pace far to fast for the NTP program to correct. The cause it was discovered was that the replacement kernel I had installed was compiled for a different processor than the model in use. The solution was to again replace the kernel with the correct version.
One of the component programs installed with NTP is ntpq[4] (standard NTP query program) which allows the system to be queried. This was the program used to allow the web interface to display the status of ntpd (the daemon) via the command `ntpq -d`, accessed via the python `subprocess` module.
As part of the requirements data logging must make use of UTC as the timezone. In every part of the system, from operating system to logging software and file data the time is set using this method, with an aim to avoid any need for conversion between local timezones and the desired output.

5.1.2 Startup

With an aim to provide a simple and easy to use solution, automating the start of the software was required, immediately after booting into the operating system. The current standard way of achieving this in Ubuntu is known as Upstart[14], which replaces the older `init` daemon within more recent versions of the distribution. Scripts were produced for each of the component processes, to start echo on boot or restart on failure.
A different solution to this problem was investigated using monit[24]. It was felt to be a more complex and unnecessary tool for the task at hand.

5.1.3 Boot Location

The Sheevaplug is capable of booting from either internal memory (NAND) or the SD card. A decision was therefore required on where to store the operating system and logging software. The entire system is small enough to fit on the 512Mb NAND, with a small amount of space left over, which would then leave the whole SD card available for data storage. Booting from the card would consume some of this available space, reducing the amount of data that could be kept. The significant advantage to using the SD card is it is removable, allowing for the simple replacement, update and duplication of the software. If the system running on the NAND was to somehow become damaged or inoperable while in use, repairing it may be difficult for end-users. Simply replacing the card with another is a much easier option.
It is important to note that the operating system is installed in both locations, on the SD card and the NAND. This allows for the SheevaPlug to be booted without mounting the card and to be used for the duplication of the contents, a process described in section 7.2.

5.2 MiniSeis

5.2.1 Introduction

MiniSeis was the chosen name for the created software and represents the core work of the project. It is written in python and relies on a few external libraries, which are detailed in section 5.2.4. The software is split into a number of processes, each of which is described in detail in the following sections of this chapter.

5.2.2 Processes

5.2.2.1 Logger

The logger process is designed to manage the connection with the seismometer, then capture data from the device, timestamp each sample and pass it onwards to the writer. The aim is to avoid doing any other work here which could delay samples being read and reduce the accuracy of the timestamps. This is the reason for splitting the logger and writer into two distinct processes, so that any I/O delays in writing to data files do not impact logging activity.
Most of the implementation is contained in a single class `MiniLogger`, though an object providing an interface to the seismometer is also employed. The obspy

function for generating UTC based timestamps is imported.
The operation of this program first attempts to find and connect to a device, followed by a main loop, which blocks for a new sample, timestamps it and places it in the queue for the writer. On disconnection an interrupted value is place in the queue and the loop is paused, returning once the seismometer is acquired again.

5.2.2.2 Writer

The process responsible for managing incoming data and writing it out to data files, both miniseed and AmaSeis, is the writer. The main body of the program is implemented in the class `MiniWriter`. The process has a main loop, which blocks waiting for new entries in the queue to appear. Each sample is then placed in a buffer object (implemented in `MiniBuffer` and is written out as needed.
For miniseed each buffer represents a new file. Files are written when the buffer size reaches a set limit for the number of samples or if an interrupt is received. In the case of an interrupt a new file is not created until the reconnection of the device, creating a gap representing in the start and end times of the new and previous file respectively. Obspy provides the functionality to write miniseed files and is employed here. For miniseed the default size of the buffers is 1800 samples or 90 second worth of data. This number was determined by experimentation as a value that would fit in a 4kb file, inclusive of headers. As compression is used, some of any file may be empty space as the miniseed format requires that the file size be a power of two.
AmaSeis files are created using the same buffer objects but with more than one buffer being used per file. Again buffers are written out to the current file as they reach a predefined size limit and as each AmaSeis file represents one hour worth of data, buffers also end when the next sample is after the switch to another hour. A new buffer created with this same next sample and the old file is completed and closed. The buffer size for AmaSeis files has a default size of 512 samples, or 25.6 seconds worth of data. This value is a trade-off between keeping the files up to date with real time data and reducing the number of writes to storage, each of which requires the locking, opening and closing of the file, as the AmaSeis server may attempt to read from the file at the same time.
An important difference between miniseed and AmaSeis is the handling of logging interruptions. In miniseed the files can be of arbitrary length as they record both the start and end times of the data they contain. AmaSeis however only records the sample data, with the hour each file represented by the file name and its location within the directory structure. Any gaps therefore are zero filled, that is a number of samples with a value of zero are written to file, based on the time of the last good sample, the time of the first sample after reconnect and the known sample rate of the seismometer. For example if the device is disconnected for

exactly a minute and the sample rate is 20Hz (SEP seismometer) then 1200 zeros are written to file before the new sample. This same function is also performed on initialisation of the process, taking into account the time since the start of the hour, or if a file for the current hour already exists, the time elapsed since the last sample.
Before writing to any file the writer asserts file based locks, releasing them when it is done, this is required to prevent the sender or AmaSeis server processes accessing the same file while it is being written. Both these processes therefore also require file locking.
The data logging activity can be disabled in the configuration, if this happens, the logger suspends its activity, waiting for the configuration to change again. When the writer encounters this scenario it flushes the remaining data to file and waits on new data appearing in the queue.
Finally the writer also performs a daily check while running of the AmaSeis log directory and removes files with an age greater than one year, a figure agreed with the BGS. It is important to note that clients which have already connected to the AmaSeis server save the data they receive locally and if they have not accessed the data for over a year it is unlikely they will do so in the future. A similar mechanism for managing miniseed files is provided by the sender.

5.2.2.3 Launcher

Startup is achieved by creating the launcher process, with the logger and writer created as a result. The communication between the logger and writer is handled with a `Queue`, part of the python `multiprocessing` module[35]. The launcher process initializes the queue, and passes it as an argument to the processes it spawns. The queue provides for straight forward interprocess communication, with the logger needing only to access the `put` method and the writer the `get` method.

5.2.2.4 Sender

The sender process is responsible for dealing with completed miniseed files, transmitting them to the server and archiving them afterwards. The directory in which the writer creates miniseed files is scanned periodically, files are read into the sender, if a lock can be acquired and transmitted in batches to the server over HTTP. On success files are moved into an intermediate directory to be archived later and on failure the files remain in place, to be retried later. The system tracks repeated failure, if the sender cannot successfully send any files for a preset number of hours it begins to archive the oldest files without further attempts to transmit them. There is continued attempt to send newer files, still within the

threshold age and when the first of these is successful the sender resumes normal operation.
Archiving is achieved via the python module `zipfile`[38] and miniseed files are written in batches to zip files, with each archive spanning one hour each. Files are grouped based on the hour in which they started, so files which cross the boundaries of each hour are contained within the earliest hour. Standard zip compression is used to reduce the storage space used by these archives, rather than simply storing them and old miniseed files are removed after archiving. Actual space used has been found to be dependent on the level of activity recorded in the files, as might be expected the smaller the difference in values, the more efficient compression is. These zipped files are made available to the end user via the web interface.
The final function of the sender is to manage the space used by the miniseed archives. When the available space on the current storage medium drops below a threshold value (default 10%) the oldest archives are removed until the free spaces reaches a slightly higher threshold (default 12%). This seeks to ensure that there will always be space for new data, without the intervention of a user required.

5.2.2.5 Web Server

The web server is a multi threaded implementation of the python module `BaseHTTPServer`[33]. A handler class `MiniSeisHandler` implements the required interface to deal with incoming requests and serve responses. Each page is generated by its own class, implementing the functions of `PageTemplate` which acts as an abstract class. Code is kept separate from html with template pages having sections replaced via a basic tagging system.
As previously mentioned one of the goals of the web server was to avoid it having to save state, in an effort to reduce resource requirements. Authentication is carried out via using the `Basic HTTP Authentication` standard, and the server checks only that a specific header value is set to the correct encoded username and password pair before allowing access.

5.2.2.6 AmaSeis Server

Access to the AmaSeis data files is provided via another multi threaded server, this time using the module `SocketServer` and with a handler implemented by the class `AmaHandler`. The implementation of this process was obviously influenced heavily by the existing AmaSeis protocol for serving files. There are however some substantial differences, relating to the handling of files, between the original and new implementations. A comment found in the original source code read

'NOTE: The AmaSeis files use the first four bytes to contain the size of the file. (I wish I had never done that.)' (wcomm.cpp). This suggested that it might be worthwhile to look at alternative ways of storing the data and transmitting it. The existing server reads these four bytes, sends their value to the client, waits for a response, then sends the rest of the data minus these bytes. The AmaSeis hour records created by the new system do not contain these four bytes and begin immediately with sample data. When the server reads a file into memory for transmission, the length of the data can be easily determined at that point and the file data sent without modification. Another difference is in the rate at which the server updates the client. The client can be set to refresh the data very quickly, providing an almost real time data feed. The default setting for this in AmaSeis is 0.2 seconds. As the writer process buffers the data samples before writing them out to file, a substantially longer period of time (25 seconds) occurs before new data is actually available. The server keeps track of the time of the last request for current data which is replied to, if the interval is too small the client is informed there is no new data yet without any file processing required.

5.2.3 Shared Components

5.2.3.1 Configuration

A number of classes are shared among the processes detailed previously. The first of these is the configuration class `MiniConfig` which allows access to read and write system settings. A helper class `MiniConfigValidator` is also used when setting new values to ensure they are valid. All processes read values from this class but only the web server, via the configuration page, ever sets values. Locking is employed again to prevent race conditions when writing new configuration data. Configuration values are stored in an XML file, which along with keys and values contains names and descriptions for each setting, allowing them to be described in a more user friendly way via the web interface. Much of the XML handling is performed via the module `xml.dom.minidom`[37].
Each process loads the configuration in different ways, the web and AmaSeis servers both load the file afresh on every new thread and so have the most up to date values available to them. The logger, writer and sender each load from file on initialisation and then periodically call the `refresh` function provided by the configuration class. This function first checks the file size and modification time versus the values recorded at the last load and only reads the file again if necessary.

5.2.3.2 Debugger

Another shared class is the debugger (`MiniDebugger`), named for its original task of assisting in debugging during development but now used for managing the system log. The name was maintained to differentiate it from the other classes, responsible for data logging. All processes append messages to the log via this class and it automatically trims the log to stop it growing beyond a reasonable size.
This log file is the one which is read for display on the status page of the web interface and it is flushed automatically on the startup of the logger process.

5.2.3.3 Others

Other shared components include a class for common utility functions and the contents of external libraries described in section 5.2.4.

5.2.4 Libraries

The following sections list and describe the usage of external libraries within the project. Licensing for these is either under the MIT license[27], for poster and lockfile, the python license[34], for pyserial or the GPL, standard[11] for the Obspy miniseed libraries and lesser[12] for the other components of Obspy.

5.2.4.1 Obspy

Obspy provides a large amount of useful functionality, split helpfully into various packages. Two of these modules were used in the final project, those being obspy.core and obspy.mseed. The parts of the core which were used was the UTCDateTime function, allowing the creation of timestamps compatible with the miniseed libraries, and Trace and Stream classes, allowing data to be formatted and written to miniseed files.
The Obspy packages are installed separately from the rest of the software, in contrast to the other libraries have been wrapped inside a module within the system called `external`. Obspy is further dependent on some additional packages, including numpy[26]. Obspy is a relatively new project but appears well tested, stable and is actively being updated.

5.2.4.2 pyserial

The pyserial package[31] provides access to serial ports and is used by the logger process to interface with the device. Specifically the device class `MiniSepDevice` handles connection management and a wrapper around the `readline` function. This module appears to be well established, stable and well tested.

5.2.4.3 lockfile

Lockfile[23] is provided in a few classes which allow the locking of resources at the file level. It is used in the writer, sender and AmaSeis server processes as well as the configuration class to prevent race conditions. When used locks are typically broken after a timeout, to cover the case where the system is powered down and a lock has not yet been released. The module has the advantage of providing multi platform support transparently, so portability is not an issue.

5.2.4.4 poster

The poster module[30] provides a library to transmit data over the HTTP POST request method. It is used in the sender process to format and transmit file data to the receiving server.Again this module appears widely used and stable.

5.3 Receiving Server

Both versions of the receiving software provide the same functionality and are capable of accepting requests to upload file from clients. All implementation is carried out using only the standard libraries for the respective programming languages with no external APIs in use.

5.3.0.5 PHP software

The PHP version of the receiving software is designed to run on a standard web server setup, such as Apache[9]. It requires only PHP version 5 to run and access to the local file system. It is implemented in four separate files. The main functionality is provided by upload.php, which accepts incoming client upload requests, authenticates them and then writes the file to disk. Management of the authorised stations list is provided via station_manager.php, which provides the ability to display, add and delete stations from the list. This list is stored as an array in station_list.php and a configuration file config.php can be edited by

hand, to set authentication details for the station manager and the directories in which to store files.
As a publicly available page, authentication with the station manager requires more security than that which is present in the client based local web server. This was achieved by using the digest based authentication provided by the standard PHP library[29].

5.3.0.6 python software

As an alternative to the PHP version of the software the python standalone web server was produced. It is similar in nature to the to web server implemented in the client but provides responses only to HTTP POST requests to the path `/upload.html`, with all other requests receiving a 404 (not found) response.
The server is again based on the module `BaseHTTPServer` and the handler is implemented in the class `Handler`. Authorised stations are maintained in a text file name `stations` and an associated script, `stations.py` provides command line management of this file. For example the command '`stations.py add STN somepassword`' is used to add a new station. As the station manager is not exposed to the internet in the same way that the PHP version might be, any security needs are met by the operating system limiting access to the script and no user authentication is required.

5.4 Portability

An effort was made to provide a portable solution, both in the MiniSeis and receiving server software. Doing so in the receiving server was relatively simple. Both versions, as previously mentioned, use only standard library functions and do not have any external dependencies, requiring only that either PHP and a web server be installed, or python.
As a more complex piece of software MiniSeis provided more of a challenge for portability. Python provides much in the way of universal access to standard operating system functions, such as handling files. Some tasks required dedicated solutions however, such as determining the available storage space remaining.
Another portability issue is that Obspy requires installation on the target hardware when setting up the system. The Obspy project though provides instructions for doing so on Linux, Windows and Mac.[5]

5.5 Extensibility

The possibility of adding futher seismomter models to the data logging software was mentioned early on the project. The current implementation includes support only for the SEP seismometer in the class `MiniSepDevice`. Adding another device would involve subclassing the existing model and overriding the functions `find_device`, `readline` and `disconnect`. The logger class `MiniLogger` would require changes to the method `__find_device_and_connect`, to alter the initialisation of the seismometer to the newly defined class or to add an option to switch between them, perhaps via the command line arguments.
Another potentially desirable area for extension is the web interface. Pages can be added by subclassing `PageTemplate` with small changes needed to `MiniSeisHandler` required to process any new path.

5.6 Testing

Unit testing was employed for some areas of the project, while some functionality was tested manually, be inspecting the output. Areas for which unit tests were produced centered around the internal aspects of the data logging process, such as configuration, sample buffering and debugging. Unit tests were produced for the two server processes as well. In the case of the web server this only tests that the server is live and that a attempt to request a page results in a 401 (authentication required) response. For the AmaSeis server the protocol itself was exercised by a basic simulation of the data a client might send when connecting and retrieving data.
The functionalities which were not subject to unit tests were those such as writing miniseed files, which produce an easily checkable output. The detailed operation of the web interface was also inspected manually. The system was left running for several days at a time while testing these items.
A total of 17 unit tests were produced for the system, which can be executed as a suite using the script `run_tests.py`. Coverage is low, due to the nature of the system and the difficulty in creating unit tests for some components.
The receiving server solutions were tested through observation and comparison of the received files to the local files held on the archives on the Sheevaplug. Authentication was tested by using both valid and invalid credentials and observing the response to the sender.

5.7 Tools Used

Subversion was used to manage the on going changes to the code base and allow work to be done from multiple locations. A simple shell script was also created for use on an internal DICE machine, to export the most recent version of the code and upload the program to the device using the tool `scp`. All code, XML and HTML was written using a text editor.
When updating and making alterations to the Sheevaplug boot process the provided installer was used, although some minor corrections were needed to get it to an operational state. More detail on this is provided in section 7.3.

6. Evaluation

6.1 Meeting

In January of this year a follow-up meeting was held with staff from the BGS, they were able to have an opportunity to view the system in an operational state (or at least the user visible parts of it) and to discuss some of the finer the details of the system, such as how to deal with old data when storage space was becoming full.
A few minor corrections were made as a result of this meeting but overall the system was thought to have met the objectives of the project.
At this point it was agreed that the receiving server software would be provided to the BGS at that time, so that they could setup their server for testing purposes. The data logging software would then be delivered a short time later, after the changes discussed were implemented and my own testing was complete. The provision of documentation for basic setup of the system was also discussed, with the BGS to produce more detailed support documents once they understood the operation of the system more fully. Details of this part of the project are discussed in section 7.

6.2 Requirements Met

All of the modified list of core requirements have been completed and in addition the optional objective of implementing the AmaSeis server has also been achieved. The system designed for the project does not implement the server side analysis component detailed in the original project specification, as discussed in section 2 this was removed as a requirement early in the project as it was felt there would not be sufficient time to complete both this and the data logging component. In hindsight this seems like the correct decision, as such a task may have taken the entire length of the project in itself.

6.3 Limitations

As previously mentioned the system interfaces only with the SEP seismometer and can produce output files in miniseed and AmaSeis formats only. The limitations are a result of what was available during development of the project and

design decisions taken, with miniseed being the format specified in the requirements. For the purposes of the School Seismology Project this situation not a problem at this point in time, but potentially will require attention if the requirements later change.
An issue which would have received more attention if there was time was the running of the MiniSeis software as the root user under Linux. Doing so is not generally seen as best practice due to potential security problems which could later be exposed but given that the system does not hold and sensitive or valuable information the risks of malicious attacks are reasonably low. Attempts were made to solve this by initialising the software with other users but a way to allow the web and AmaSeis servers to bind to their required ports was not found without the use of root access.

6.4 Abandoned Features

Some experimentation was done with the Obspy imaging library in an effort to produce plots of the most recent data for the local web interface. This was abandoned in favour of spending the time ensuring the core components were tested and working properly. This functionality is also likely to be among the first things the BGS seeks to do with the data when it arrives at the server and so as duplicated functionality it may not have much value, especially is the server side solution allows for more control over any analysis.

6.5 Problems Encountered

In the early stages of the project a number of technical issues occurred which slowed progress. With the Sheevaplug ordered but yet to arrive a DICE machine was used in its place to interface with the seismometer. Unfortunately by default permissions are quite restrictive on DICE accounts and it took some time to gain access to the serial port. Once this was fixed the lack of documentation for the analogue to digital converter, used between the seismometer and the computer, caused more problems. Significant time had to be spent experimenting with different connection parameters, such as baud rate and parity, before useful data could be retrieved.
Setting up the Sheevaplug on the Informatics network also took a little longer than was hoped. Moving it for the January demonstration, to another room caused it to lose access to the network as it had only been setup for use on a specific port. At one point the power connector within the Sheevaplug disappeared into the body of the device, caused by missing screws which meant to hold it in

place. Replacement screws were quickly sent on by the supplier.

6.6 Resource Usage

One of the original project goals was low key data logging on a low powered device, which is interpreted to mean that minimal CPU and memory resources should be consumed by any solution. The complete solution runs easily within the resources provided on the standard Sheevaplug, a 1.2GHz single core CPU and 512Mb of RAM. The most CPU intensive processes are the logger and writer, at a around a total of 4% (constantly) between them, with the others using very little and in infrequent bursts. The total memory footprint for all processes while running is around 10.2% or 52.2Mb.
It would be fair to say that the resource requirements could have been reduced through the use of an alternative language, such as C or by combining all functionality into a single, multithreaded process, to reduce overheads. Given the ample resources available and the likelihood that any replacement for the Sheevaplug in years to come will be more powerful rather than less, the current usage does not appear to be any cause for concern.

7. Delivery

7.1 Documentation

As part of the January meeting is was agreed that I would produce some basic documentation to guide the BGS through setup and use of the system, as well provide a basis for the final end user documentation. Several different sets of instructions were produced, in different formats.
At the end-user level, help topics were added to the web interface homepage, which give instructions on how to access and use each part of the interface to manage the device. A pdf file containing setup instructions for connecting the Sheevaplug to the seismometer and network was also created.
A further two documents were produced for the benefit of the BGS or others who might work on the system in the future. The first of these details the duplication of the contents of the SD card using the Linux command `dd` and a USB drive with the Sheevaplug. The second document explains the use of the installer tool, how to modify it to get it operational and create or recover a cleanly installed Sheevaplug, by replacing both the uboot and kernel binaries.

7.2 Duplication

As the Sheevaplug has a USB port, it can be used to provide additional storage to perform the duplication of the SD card, without requiring another PC. Instructions for doing this were provided to the BGS as well as the image of the final version of the software from the project.
In brief the following steps need to be taken to create an image of the system.

1. Format a USB drive with the FAT32 filesystem.
2. Boot the Sheevaplug from the NAND based kernel.
3. Insert and mount the USB drive.
4. Insert the SD card.
5. Create an image using the command `dd if=/dev/mmcblk0 of=/mnt/usb/file.img`

The reverse, writing the image to a new card, requires only that the arguments for the last instruction be reversed.

7.3 Installer

The available installer for the SheevaPlug does not appear to work as it is provided. Some basic changes to the script that runs it and an update of the binaries which it places on the device were required to make it useful again. Instructions for using the installer and repairing it if needed are provided to the BGS, along with my own copy of the installer. The fixed version was only tested and run on Linux, the Windows version may or may not work. The installer is available on the provided development CD or from the internet[7].

8. Further Work

8.1 BGS Usage of Collected Data

Waiting for information...

8.2 Extensions to the System

The project provides a strong basis for further efforts which make use of the data, once it is available on the receiving server. Among these could be the original project objective of providing remote analysis of individual files via a web browser. Allowing direct access to logged data may also be desirable and converting and making available the miniseed data in other formats may also be of use to end users.
There are also some possible alterations that could be made on the data logging side which would allow for more functionality. Adding support for additional seismometers would be an obvious first step. The Sheevaplug CPU and memory resources also allow for considerably more intensive processing to take place locally if desired for example analysis of recorded data and image plotting. Extending the logging functionality to provide different file formats as output may also be desirable, if data were to be collected for another purpose.

8.3 AmaSeis Replacement

Another project seeking to replace AmaSeis on a more complete level is jAmaSeis[15], a Java based application which allows for significantly easier sharing than the current AmaSeis client/server model and adds new functionality for analysis of recorded data. The software is not yet available and is part of a project which has seen computer science students participate in its development at Moravian College[16]
Another extension to project detailed in this report stems from jAmaSeis, that of allowing the new clients access to recorded data over the internet.

Bibliography

[1] British geological survey. http://www.bgs.ac.uk/.

[2] Iris website. http://www.iris.edu/hq/.

[3] Newit. http://newit.co.uk/.

[4] ntpq. http://www.eecis.udel.edu/~mills/ntp/html/ntpq.html.

[5] Obspy. http://obspy.org.

[6] Rapid. http://research.nesc.ac.uk/rapid/.

[7] Sheeva plug distributions. http://sheeva.with-linux.com/sheeva/.

[8] Ubuntu. http://www.ubuntu.com/.

[9] Apache. http://www.apache.org/.

[10] Buffalo. Nas catalogue. http://www.buffalotech.com/products/network-storage/.

[11] Free Software Foundation. Gpl v3. http://www.gnu.org/licenses/gpl.html.

[12] Free Software Foundation. Lesser gpl v3. http://www.gnu.org/licenses/lgpl.html.

[13] IETF Network Working Group. Rfc 2617. http://www.ietf.org/rfc/rfc2617.txt.

[14] IETF Network Working Group. Rfc 2617. http://upstart.ubuntu.com/getting-started.html.

[15] IRIS. jamaseis annoucement. http://www.iris.edu/hq/programs/education_and_outreach/software/jamasei%s.

[16] IRIS. jamaseis project. http://www.iris.iris.edu/10_Proposal/PreViews/168.html.

[17] IRIS. Manuals page. http://www.iris.edu/manuals/.

[18] IRIS. miniseed file format. http://www.iris.edu/manuals/SEED_appG.htm.

[19] IRIS. Sac format page. http://www.iris.edu/software/sac/manual/file_format.html.

[20] IRIS. Seed file format. http://www.iris.edu/manuals/SEEDManual_V2.4.pdf.

[21] IRIS. Seismographs in schools project. http://www.iris.iris.edu/hq/sis.

[22] A Jones. Amaseis website. http://pods.binghamton.edu/~ajones/AmaSeis.html.

[23] lockfile module. http://pypi.python.org/pypi/lockfile.

[24] Monit. http://mmonit.com/monit/.

[25] MUTR. Sep manual. http://www.mutr.co.uk/images/Seismometer.pdf.

[26] numpy. http://numpy.scipy.org/.

[27] opensource.org. Mit license. http://www.opensource.org/licenses/mit-license.php.

[28] php website. http://www.php.net/.

[29] php.net. Http authentication. http://php.net/manual/en/features.http-auth.php.

[30] poster module. http://pypi.python.org/pypi/poster/0.8.0.

[31] pyserial module. http://pyserial.sourceforge.net/.

[32] python. http://www.python.org.

[33] python.org. basehttpserver module. http://docs.python.org/library/basehttpserver.html.

[34] python.org. license page. http://docs.python.org/license.html.

[35] python.org. multiprocessing module. http://docs.python.org/library/multiprocessing.html.

[36] python.org. Pep8. http://www.python.org/dev/peps/pep-0008/.

[37] python.org. xml minidom module. http://docs.python.org/library/xml.dom.minidom.html.

[38] python.org. zipfile module. http://docs.python.org/library/zipfile.html.

[39] British Geological Survey. School seismology project website. http://www.bgs.ac.uk/schoolSeismology/app/schoolSeismology.cfc?method=v%iewLatestQuake.

[40] Global Scale Technologies. Sheevaplug. http://www.globalscaletechnologies.com/c-6-sheevaplugs.aspx.

[41] Jano van Hemert. Project specifcation. `http://research.nesc.ac.uk/node/499`.

[42] Wikipedia. Coordinated universal time. `http://en.wikipedia.org/wiki/Coordinated_Universal_Time`.

[43] Wikipedia. md5. `http://en.wikipedia.org/wiki/MD5`.

[44] Wikipedia. Network time protocol. `http://en.wikipedia.org/wiki/Network_Time_Protocol`.

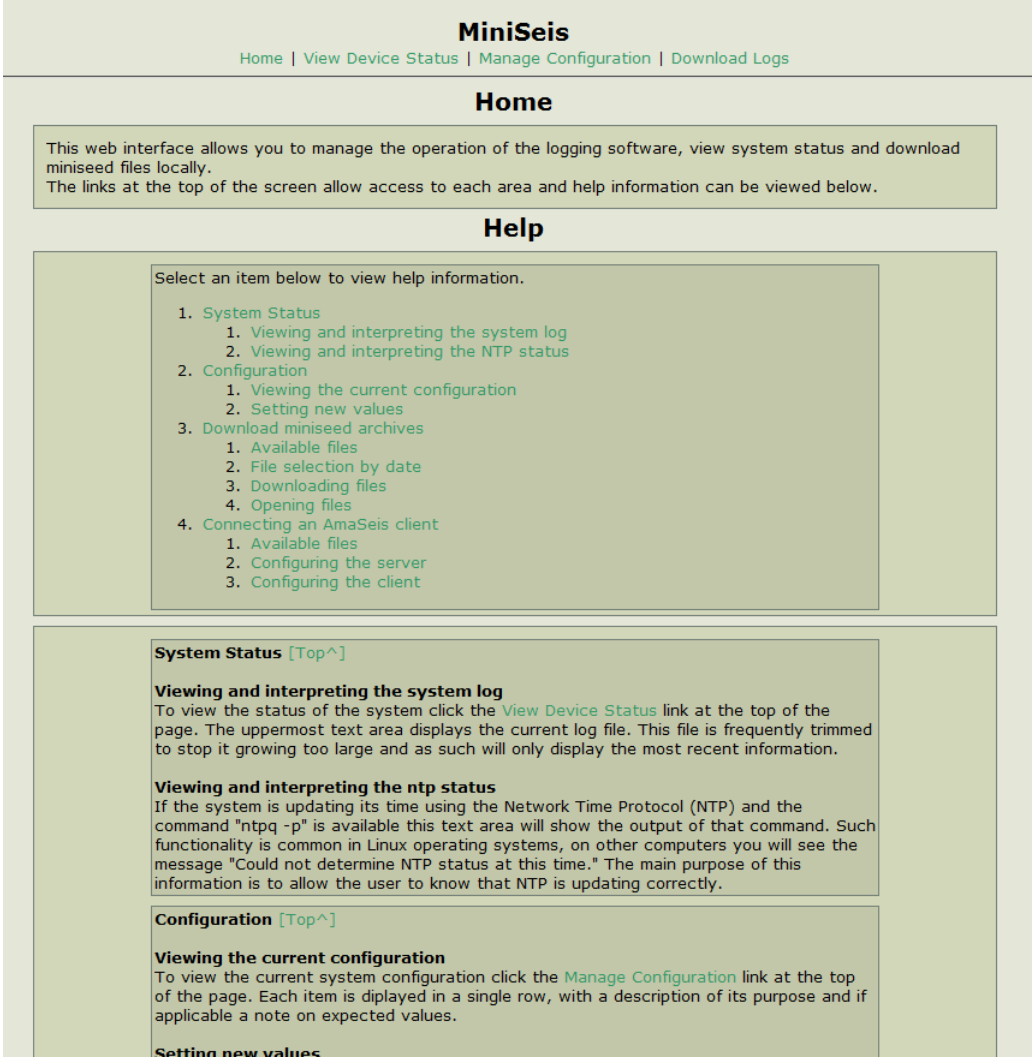
[45] Wikipedia. Nslu2 article. `http://en.wikipedia.org/wiki/NSLU2`.

Appendices

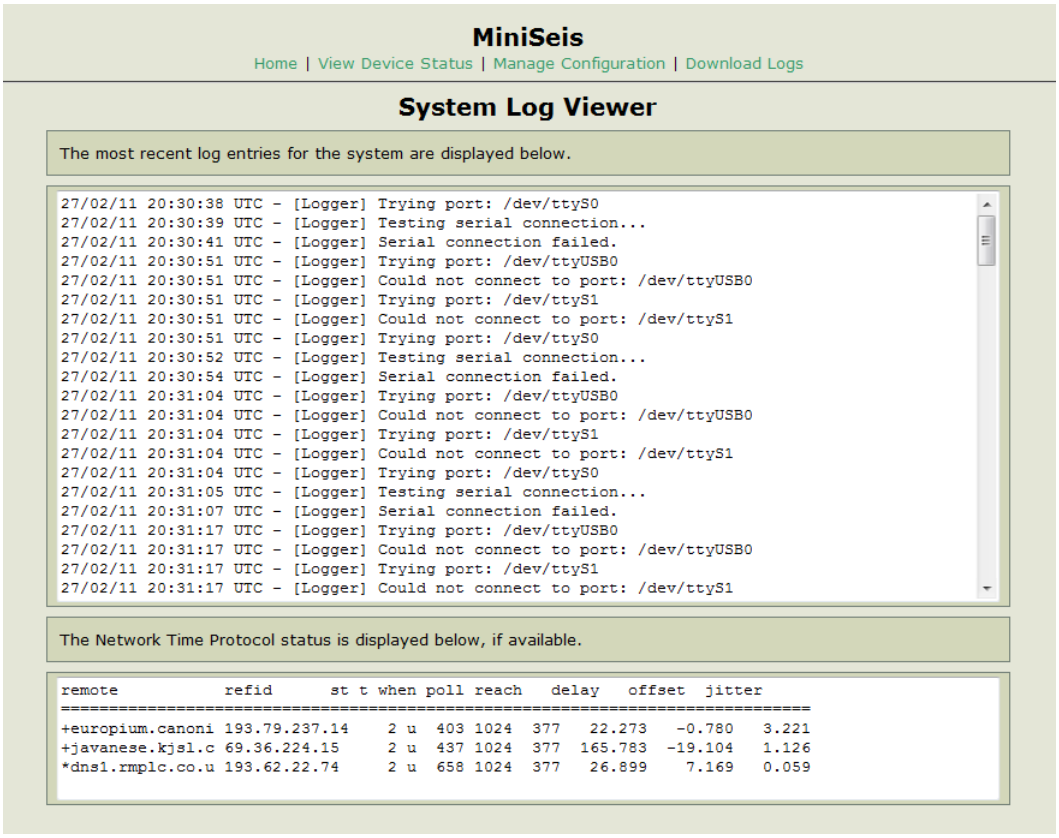
Appendix A. Web Interface

A.1 Screenshots

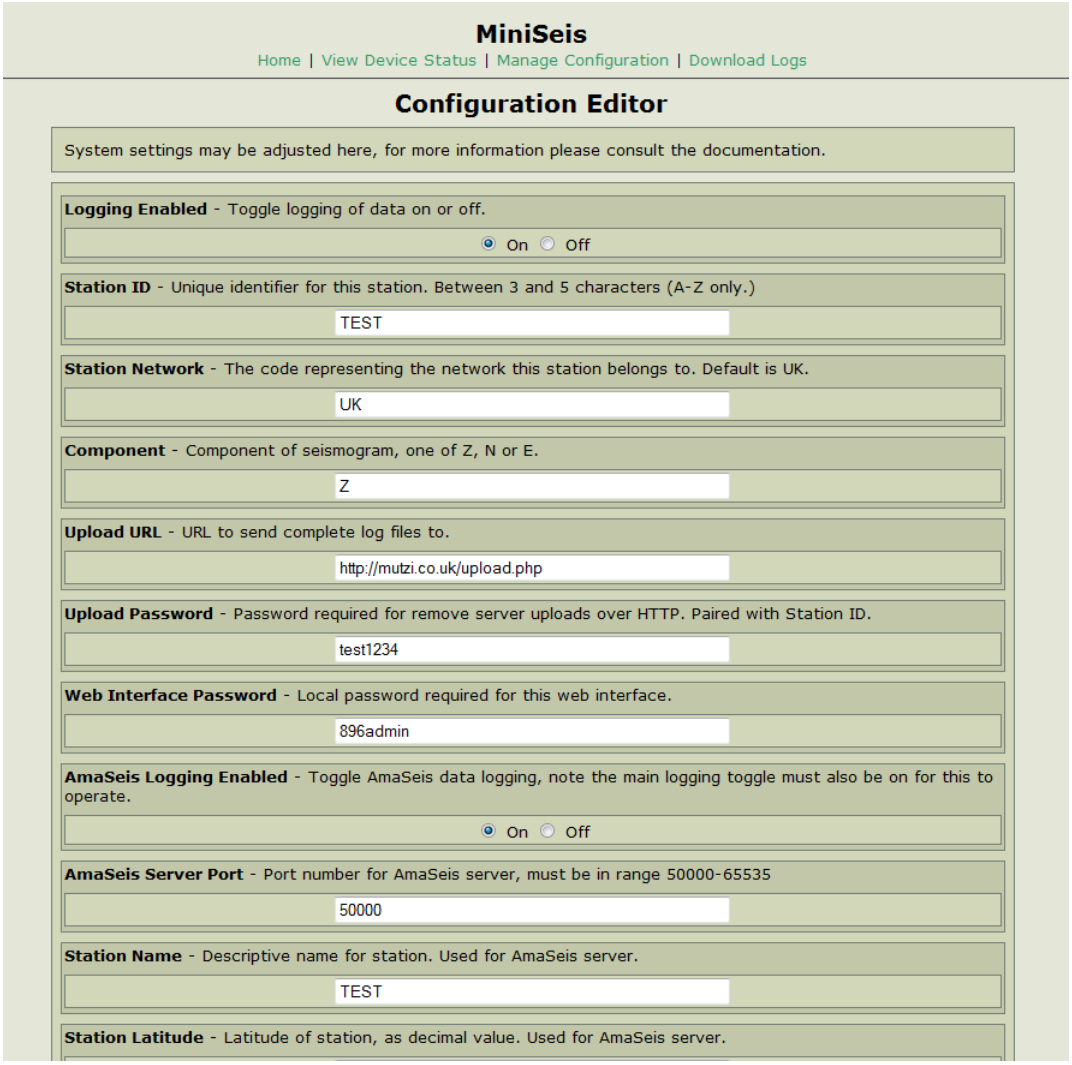
A.1.1 Help



A.1.2 Status


MiniSeis
Home | View Device Status | Manage Configuration | Download Logs
System Log Viewer
The most recent log entries for the system are displayed below.
27/02/11 20:30:38 UTC - [Logger] Trying port: /dev/ttyS0
27/02/11 20:30:39 UTC - [Logger] Testing serial connection...
27/02/11 20:30:41 UTC - [Logger] Serial connection failed.
27/02/11 20:30:51 UTC - [Logger] Trying port: /dev/ttyUSB0
27/02/11 20:30:51 UTC - [Logger] Could not connect to port: /dev/ttyUSB0
27/02/11 20:30:51 UTC - [Logger] Trying port: /dev/ttyS1
27/02/11 20:30:51 UTC - [Logger] Could not connect to port: /dev/ttyS1
27/02/11 20:30:51 UTC - [Logger] Trying port: /dev/ttyS0
27/02/11 20:30:52 UTC - [Logger] Testing serial connection...
27/02/11 20:30:54 UTC - [Logger] Serial connection failed.
27/02/11 20:31:04 UTC - [Logger] Trying port: /dev/ttyUSB0
27/02/11 20:31:04 UTC - [Logger] Could not connect to port: /dev/ttyUSB0
27/02/11 20:31:04 UTC - [Logger] Trying port: /dev/ttyS1
27/02/11 20:31:04 UTC - [Logger] Could not connect to port: /dev/ttyS1
27/02/11 20:31:04 UTC - [Logger] Trying port: /dev/ttyS0
27/02/11 20:31:05 UTC - [Logger] Testing serial connection...
27/02/11 20:31:07 UTC - [Logger] Serial connection failed.
27/02/11 20:31:17 UTC - [Logger] Trying port: /dev/ttyUSB0
27/02/11 20:31:17 UTC - [Logger] Could not connect to port: /dev/ttyUSB0
27/02/11 20:31:17 UTC - [Logger] Trying port: /dev/ttyS1
27/02/11 20:31:17 UTC - [Logger] Could not connect to port: /dev/ttyS1
The Network Time Protocol status is displayed below, if available.
remote refid st t when poll reach delay offset jitter
==
+europium.canoni 193.79.237.14 2 u 403 1024 377 22.273 -0.780 3.221
+javanese.kjsl.c 69.36.224.15 2 u 437 1024 377 165.783 -19.104 1.126
*dns1.rmplc.co.u 193.62.22.74 2 u 658 1024 377 26.899 7.169 0.059

A.1.3 Configuration



A.1.4 Archive Access