A Linux-based graphical user interface for the creation and reading of the frame format: Version 2.4

Muhammed Aoun
Edith Cowan University

Honours Research Thesis

"A Linux-based Graphical User Interface for the Creation and Reading of the Frame Format"

Version 2.4

Author: Muhammed Aoun
Supervisor: Dr Dong Li
Student #: 0983222
Date: February 2003

STANDARD DECLARATION

I certify that this thesis does not, to the best of my knowledge and belief:

(i) incorporate without acknowledgment any material previously submitted for a degree or diploma in any institution of higher education;

(ii) contain any material previously published or written by another person except where due reference is made in the text; or

(iii) contain any defamatory material

5/3/03

Contents

1.0 Abstract

To gain further knowledge of how the universe began, a new international co-operated project wishes not only to "see" stars exploding, but also to "hear" the explosion. It is theorised that the ability to "listen" to an explosion is possible by detecting gravitational waves. Gravitational waves are omitted when a star explodes, travelling through the cosmos at an exceptional speed. It is estimated that a gravity wave takes only 40 milliseconds to pass through the earth. Thus the aim of this international collaboration is to detect a gravitational wave.

To accurately detect such waves, there needs to be several global detectors. If a phenomenon is detected at one detector, data from other detectors is compared; to investigate whether this event occurred simultaneously around the world. An integral component of gravitational wave detection includes the real-time saving of signals as data on a generic storage media, e.g. a standard hard disk and later permanent storage e.g. on compact discs.

It is therefore crucial that there is a constant 24-hour data gathering from the detectors. 24-hours of constant data gathering requires a system that can facilitate for the acquisition of the data and save the data into a standard format, which was agreed upon by all the global gravity wave detection sites (known as the frame format. Please refer to appendix 9.1 for further details concerning this format.). This accommodates the exchange of the data for analysis between detection sites.

To create exchangeable files in the frame format requires the data to be passed through C Unix based libraries. These libraries are cumbersome to utilize, require sufficient computing skills to use and are very inefficient when working with large amounts of data.

The purpose of this honours project is to research and construct a graphical user interface under the Linux Operating System, to the frame libraries, in order to greatly assist in the data storage and exchange process.

2.0 Figures and Tables Overview

The figures and tables numbering convention, begin with the section the table or figure is located in, followed by an index number.

2.1 List of Figures

2.2 List of Tables

3.0 Introduction

This document will endeavour to cover the scope of the honours research that shall take place. It will cover such areas as the background of the study, its significance and its purpose.

Whilst adhering to the traditional requirements of a "thesis document", this paper also explores a practical methodology of developing an application which meets the criteria requested by ACIGA (Australian Consortium for Interferometric Gravitational Astronomy), i.e. simply put, construction of a Graphical User Interface to the frame manipulation libraries.

3.1 Background

Significant achievements have been made in the field of space exploration. It is now possible to observe and catch glimpses of distant stars and planets through such instruments as the Hubble Telescope and similarly, it is possible to witness these as they explode, giving scientists an insight of how the universe began.

Albert Einstein had theorised the existence of gravity waves. When a star explodes, it is believed that a wave of gravity is omitted. This gravity wave is said to travel at the speed of light sweeping across the cosmos, including travelling through the earth. These waves are so tiny however, that the human being cannot sense their existence. To detect these waves, a gravity wave detector must be capable of detecting a pin drop on the other side of the earth.

Gravitational wave detectors have been under development for over 30 years. Increased funding in the mid 90's has opened the gate for a new generation of gravitational wave detectors in the form of Interferometers. Four major global locations are currently developing the detectors. They are listed in the Appendix, Section 9.5.

These detectors are located in specific locations to allow the best detection. It was important to include a detector in the southern hemisphere, thus the location of the AIGO site near Gingin.
Note: It is beyond the scope of this document to discuss in detail the Laser Interferometers. References are made however, to relative literature.

The AIGO detector site remains under the supervision of the physics department of the University of Western Australia.

3.2 Significance and Purpose

The Interferometers, once functioning, will be "online" constantly, thus continues data gathering is essential. Seismic noise such as wind, voices or even moving people or cars may effect the detection of gravity waves due to their sheer frailty. Therefore as a basic necessity, all these noise omitting elements must be counted for and factored out by the detectors. Various instruments are used to sense these seismic activities. The data from these instruments also must be saved in the frame format for any future inspection. Similarly, seismic activity may occur at any time. Consequently, continuous data gathering is also essential.

The real-time data acquisition from the interferometer and seismic instruments are time stamped with a GPS signal. Let us assume some detection occurred through the Interferometer. Data gathered at the same time as the detection, from the seismic instruments, will be analysed to notice whether seismic activity occurred at the same time, meaning that the Interferometric reading likely occurred due to seismic noise.

For example, a phenomenon was discovered in one detector, say in Japan. The Japanese site wishes to investigate if any simultaneous phenomena occurred at the other detection sites. It then requests to have all data gathered during the particular time the phenomena occurred. The data stored at the other detection sites cannot be raw. Having so opens the door to ambiguity and confusion concerning what the data represents, what time it was taken, etc.

As has previously been mentioned, several detectors exist globally, with each detector requiring constant data acquisition from various sources. This constant gathering builds up to large amounts of data per day.

Personal size approximation of gathered data depends on several factors. These are based on:

- Which instrument is used to gather data? (Refer to appendix 9.4 for an overview of data gathering instrumentation)
- The sample rate at which the instruments are gathering data.
- The timing at which the data is gathered (e.g. perhaps 5 minutes in every hour).
- The number of channels attributed to the instrument.

We can get a brief insight to the expected amount of data to be gathered by doing a simple calculation based on an anemometer (wind speed and direction measuring instrument). Although the significance behind the use of the anemometer for the ACIGA project will not be explored, it is enough to say wind may affect the integrity of gravitational wave detection, and therefore wind speed and direction measurements are essential.

A preliminary test carried out by the ACIGA group has found the anemometer has sampling rate of 1024 hertz, with a sampling depth of 2 bits (1 pulse (bit) for wind speed information and the other for wind direction). 1024 * 2 * 60 (seconds) * 60 (for an hour) * 24 (for a day) = 21 megabytes of data.

21 Mbytes is quite large. Remembering this data only comes from one instrument, with 2 channels, at a medium sampling rate, for 1 day worth of data. The ACIGA project management have quoted LIGO (detector site located in the United States) project team as saying they gather approximately 1 Terabyte of data per week. It is quite phenomenal.

The "frame format" was agreed upon by all project members as the format in which data is stored. It requires various pieces of information to be supplied within the frames (see appendix 5.1 for frame format). This includes the source of the data, GPS timing, channel names, and various parts of information depending on the nature of the data.

LIGO has produced UNIX command based C programs to save data into the frame format. The ACIGA project has not had the capability and know-how thus far to process raw data into the frame format, mainly due to the expertise required to work under a UNIX environment and the mere cumbersome development method of the frame making programs.

The nature of these programs is command line based. If several files of data are required for processing, 1 command line has to be manually inputted for every file that is to be processed. This can be a gruelling task. LIGO stores their data in raw format and not in frame format. Any data which is required to be exchanged with other detector sites is then converted to frames. However, little data exchange at the moment is taking place due to the fact that the global gravity wave detectors are yet to be up and running, thus the urgency for more efficient data exchange methodologies are not required in the present, but sometime in the future.

ACIGA wishes to store all their gathered data in the frame format. The rational behind this is, if any other detector site requires a piece of data, the data will be available online, with special access privileges to the other detector sites. This saves the hassle of locating the data, compressing and converting it to frame files, then sending the file to its destination.

The ACIGA project requires a user friendly, Linux Graphical User Interface (GUI) application to cater for frame composition from raw data, as well as reading the data within frame files. The application is to allow automated multi-conversion capabilities and to limit the repetitive manual conversion requirements when using the LIGO frame libraries.

3.3 Research Question

Several research questions exist concerning the methodological approach to developing the application.

1. Frame libraries
    a. How much information on the frame libraries is available and is it freely accessible?
    b. Do the frame libraries allow foreign application interactivity?

2. Programming language
    a. Which is the most appropriate language to use for development?
    b. Does the language facilitate foreign library interactivity and is it the most efficient to use?
    c. Does the language facilitate simplified graphical interface construction?

3. Design specifics.
    a. How will training in using the application take place?
    b. What support features will be available to assist in using the application?
    c. How will the application be constructed in such a way, that its ease of use is maximised, whilst addressing the required functionality?

4. Efficiency metrics
    a. What will be the improved speed of execution of conversion of frames files?
    b. How will a feasibility study be carried out to ensure an accurate cost/benefit analysis?

5. Addressing the above queries.
    a. Who needs to be interviewed and what needs to be asked?

3.4 Previous Studies in this Field.

Three fundamental elements form the basis of the studies in this field.

1. Studies into Linux GUI development.
2. Studies into the frame format and libraries.
3. Studies into constructing a Linux based GUI for the frame libraries.

Studies in Linux GUI interfaces are no doubt large. The sheer development of tools and applications are witness to such studies. Linux, which is based on UNIX, is very much a command line operating system and "is essentially faceless. The child of the console generation, Unix began to develop into a life of its own back in the late 1970's on text-only, teletype and CRT oriented systems that lacked anything remotely resembling graphical user interfaces."[1]

In contrast however, studies into the frame format and the frame libraries would be sparse and extremely limited. Simple reason is due to the fact that the frame format is of little use to anyone outside the gravitational wave detection project.

This leads to the third and most important element, i.e. studies into GUI development for the frame libraries. Such a study would undoubtedly be confined to any of the gravitational wave project groups, since they are the only individuals with any interest in the frame libraries.

Naturally, close collaboration takes place between the various gravitational wave detection groups. Contacts with VIRGO and LIGO developers of the frame libraries have suggested no study into an interface for the frame libraries currently exists, however it was to be considered in the future.

[1] Hammel, M.J. (2001) Linux Magazine: The History of XFree86 - Over a Decade of Development

4.0 Literature Review

Answering:

How much information on the frame libraries is available and is it freely accessible?

Which is the most appropriate language to use for development?

Does the language facilitate simplified graphical interface construction?

An investigation of currently available literature related to the frame format is found in this section. The frame format was created specifically to suite the needs of the gravitational wave detection project and thus its use is limited to the domain of the project participators. Naturally, due to this limited use, very little documentation and knowledge base is attributed to this format, basically due to the fact that it is a virtually useless tool for those beyond the scope of the project.

(VIR-MAN-LAP-5400-103, 2002) and (LIGO-T970130-D-E, 2000) are the only two formal documentations published in relation to the frame format, its specifications and know-how of the use of its libraries. Both texts pay specific details to the make up of the libraries and explicit descriptions concerning variable attributes. Therefore it is without choice that frame related literature review be bound by these two documents.

The same must be said concerning QT, the GUI development environment. "There are only 2-3 books regarding Qt, the popular GUI C++ Toolkit for UNIX, Windows, MacOS and embedded Linux. Without doubt, the most important and up-to-date book for Qt until now is "Programming with QT, 2nd Edition", by Matthias Kalle Dalheimer, published by O'Reilly."[2]

It is for this reason, virtually all well known literature concerning the frame format, and the QT development environment, will be reviewed. Reasons as to why QT was the choice of development environment shall also be discussed.

[2] Queru, E. (2002) Book Review: Programming with Qt, 2nd Edition:

4.1 Specification of a Common Data Frame Format for Interferometric Gravitational Wave Detectors (IGWD)

Literature title: Specification of a Common Data Frame Format for Interferometric Gravitational Wave Detectors (IGWD)

Code and Version: VIRGO-SPE-LAP-5400-102 December 2000

Available online at: http://wwwlapp.in2p3.fr/virgo/FrameL/T970130-D-E.pdf

Description - Summary

"This specification formally defines the IGWD Frame for data structuring and exchange that is to be used where applicable"

This document deals intensely with the makeup of the Frame file structure. Each of the (following) structures is defined, explained and its data types are listed.
"A Frame is a grouping of multiple C structures composed of the following elements:

- Frame header
- Dictionaries permitting reconstruction of the C structures via reading of frame data off media
- Frame history comment
- Detector/instrumental configuration
- Raw fast data
- Serial data
- Event trigger data
- Post-processed/derived data
- Simulated data"

Relevance - Evaluation and Analysis

Whilst detailed and technical explanations are given to the frame file format, the question remains, how or more specifically, does this relate to the application development. The make up of the frame files are clearly described in this document. Any individual with medium C language knowledge may find understanding the document straight forward.

Little mention, however, is made to instructions or examples of how to use the libraries to create frames, remembering that it is not the actual conversion process that which the application development is mainly concerned with, but will the libraries allow to be executed by a foreign source. Where relevance to the study is prevalent may exist, is in the actual makeup of the frame files. It is required the GUI interface have the ability to structure the frame files in specification to this literature. Whether it does or not depends on the subsequent research.

4.2 Frame Library (Fr) User's Manual

Literature title: Frame Library (Fr) User's Manual

Code and Version: VIR-MAN-LAP-5400-103 Version 6.02 October 22, 2002

Available online at: http://wwwlapp.in2p3.fr/virgo/FrameL/FrDoc.html

Description - Summary

This literature really goes "hand in hand" so to speak, with the above literature. Whilst the above document deals with the specification of the construction of the frame file, this document deals with the specifications of the construction of the frame libraries, making reference to the frame construct where possible. "The data are stored in a set of C structures described in the document Specification of a Common Data Frame Format for Interferometric Gravitational Wave Detector (IGWD) (VIR-SPE-LAP-5400-102 and LIGO-T970130-E). The variable names of the C structures are exactly the one given in this document." (VIR MAN LAP-5400-103, 2002).

Here, the various libraries available are listed, and each one of them is described. Whilst not giving the full code of the libraries themselves, what is described is the Input and Output of variables to the C structures. An example of one such library is FrCheck (i.e. frame check). This program will do a check on the converted frame file to see whether it can be read successfully, meaning it was converted successfully. (see section 5.3.1 for further details concerning FrCheck)

Relevance - Evaluation and Analysis

This document includes the very basis of the instructions needed to know how to run the frame libraries, clearly signifying the importance to the application development. Let us revisit the FrCheck library, the following reference is made:

The syntax is: FrCheck *options*

where *option* could be:

-i <input file> Only one file should be used
-d <debug level> (default 1). 0 will supress all info and error messages.
-t to scan the file using only the TOC
-s to scan the file using only sequentiel read(TOC not used)
-f GPS time of the first frame to scan (default=0) (only used when doing the random access)
-l GPS time of the last frame to scan (default : 999999999.) (only used when doing the random access).
-h to get the help

Using a command under Linux, and from the above information, we may do a basic check a file similar to the following manner:

```
$: FrCheck –I nameoffile
```

Knowledge of the options of the libraries is essential, for such options have to be implemented in the GUI and automated, to save the hassle, for e.g. of doing the above line multiple times.

4.3 QT

Literature title: Qt Reference Documentation

Version: Qt version 3.1.0

Available online at: http://doc.trolltech.com/3.1/

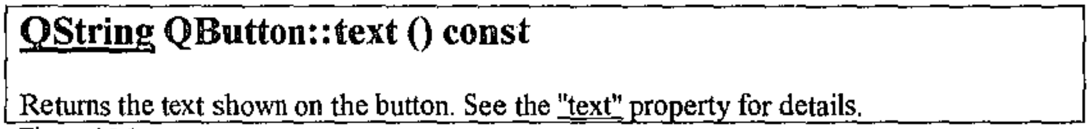
Description - Summary

The most comprehensive documentation attributed to the QT development environment no doubt comes from Trolltech, the developers of QT. The reference documentation is broken up into several major parts, i.e. Getting started, API Reference and Tools.

Relevance - Evaluation and Analysis

Getting started documentation is excellent in the way it assists new users of QT. Whitepapers are available which give an overview of the whole QT environment. The most helpful characteristics however, is the vast number of coding examples that's included with the literature. "Qt ships with lots of small and some medium-sized example programs that teach you how to implement various tasks with Qt." "Lots" means approximately 60. These cover all major aspects from code, to widgets and to using the various tools.

Perhaps more importantly from the developers point of view is the API reference documentation. Every single QT class and function is given, explained and links are available to related subjects. For example[3]:



QString QButton::text () const

Returns the text shown on the button. See the "text" property for details.

Figure 4.3.1

The above reference implies that if one wishes to return the text property of a QButton, they call the text() function which is a QString (i.e. string) variable, in the following manner

QString buttonText = new my_button->**text();**

[3] Trolltech Technologies. (2002) QT Reference Documentation

Literature title: Programming with QT: Writing Portable GUI Applications on Unix & Win 32 - 2nd Edition

Publication:

Author: Matthias Kalle Dalheimer

Description - Summary

"The book is one of the richest and most complete guides of any GUI Toolkit books I read so far. It starts with a really good introduction for Qt, and also pinpoints the reasons why Qt is using GUI emulation and not API emulation (Motif, MFC) or API layering (wxWindows). Quite some food for thought, really."[4]

Dalheimer explains, "other products allow portable programming for Unix and MS Windows. These products include commercial libraries like Zinc and free libraries like wxWindows. I have evaluated most of them for my projects and have always found QT to be the best option"[5]

Relevance - Evaluation and Analysis

Of particular importance in this book, Dalheimer discusses child processes spawning from the application. "With QProcess, you spawn a process, write to the standard input channel of this process, read from its standard output and standard error channels, and are notified when the process has finished working."[6]

Just as important is the example which is given, giving the reader a greater insight in how QT works.

```
Process = new QProcess();
Process->addArgument( "ls");
Process->start();
```

This element fill form the base ability in which the GUI can communicate with the frame libraries.

[4] Queru, E. (2002) Book Review: Programming with Qt, 2nd Edition
[5] Dalheimer, M.K. (2002) Programming with QT 2nd Edition : O'Rielly
[6] Dalheimer, M.K. (2002) Programming with QT 2nd Edition : O'Rielly

5.0 Requirements and Materials

Answering:

Does the language facilitate foreign library interactivity and is it the most efficient to use?

How will the application be constructed in such a way, that its ease of use is maximised, whilst addressing the required functionality?

This section explores answers to the research questions. It describes the approach which data needs to be gathered concerning the design and implementation of application, as well as what tools are to be used to gather this information.

5.1 Requirements

Design considerations include the following investigations:

- Interface design - How is the application to look like. Such requirements are solely to the preferences of the ACIGA group.
- Range of functionality — is the application limited to only interface to the frame libraries.

These considerations will take place through interviews. Such information need only extracted through low-level informal consultation with the target population.

Each consideration shall be evaluated in terms of time / efficiency of implementation. Time to collect such information is not extensive.

Understanding of the frame libraries takes the bulk of the data gathering; this consists of understanding the frame structure and the frame libraries. Such information is gathered through relevant literature reviews and specifications.

5.1.1 Target Population

The only populaces to benefit from any data gathered, relative to gravitational wave detection, are those directly or indirectly related to the project. These include the TAMA, LIGO and VIRGO projects and their individual subordinates. Please refer to Section for further information concerning these projects.

As previously mentioned, all three have agreed upon that the standard data exchange format between the detection sites be that of the frame format.

The ACIGA project is the main point of concern. It will be the sole user of any developed application.

5.1.2 Research Questions

The primary objective of the research, which must be met to satisfy the thesis, is that of understanding the Frame Libraries and its construction.

To do so, a beneficial method would be that similar to experimental action research. "Action Research is a fancy way of saying let's study what's happening at our school and decide how to make it a better place" (Emily Calhoun, 1994). In this situation we really would say "let's study what's happening with the frame libraries and decide how to make their use much better"

The research questions involve several elements that need to be addressed for the creation and testing of the application.

Frame libraries

How much information on the frame libraries is available and is it freely accessible?
The best approach to answer this question is basically to do more researching, with the most efficient method being to contact LIGO and VIRGO directly and asking whether there are other then the two literatures mentioned above relating to frames.

Do the frame libraries allow foreign application interactivity?
A study of the libraries themselves would be the best procedure here. Obtaining the libraries (freely downloadable from the WWW) and creating a small testing application to check whether interactivity is possible.

Programming language

Which is the most appropriate language to use for development?
One criterion must be met here. This is whether the language can be compiled under the Linux Operating System environment. If so, the next step is to estimate the easy of use of the language, for ease of use speeds up production. One critical element is the ease of creating a GUI. The selection of such languages is not wide, with ANSI C++ being the most likely option. An exploration of the various languages will take place. Fortunately most language compilers found under the Linux OS are open-source and freely available for download.

Does the language facilitate foreign library interactivity and is it the most efficient to use?
Again, this needs to be addressed by exploring the language itself.

Does the language facilitate simplified graphical interface construction?
It is possible to create a GUI using the assembly language, but naturally this 1st generation language would be a gruesome approach for such a task. Perhaps the real question here is how easy is it to construct a GUI using the selected language,

remembering such languages as C++ offer object libraries, where GUI objects such as windows and menus are already available for use.

Design specifics

How will training in using the application take place?

Remembering that the application is to be used mainly by physicists, not all are completely computer literate. Perhaps the easiest approach here is to simply ask the target population if training is required after presenting them with the finished product.

How will the application be constructed in such a way, that its ease of use is maximised, whilst addressing the required functionality?

Three issues need to be addressed here, firstly the actual application development. The libraries may well contain a large array of functionality, with much that may or may not be needed to be used by the ACIGA group. There is constant consultation with individuals in the ACIGA group; the answer to what functions are needed may be addressed by them. However, the available functions themselves must be obtained; this is through the exploration of the frame libraries.

Efficiency metrics

What will be the improved speed of execution of conversion of frames files?

Arbitrary data will be created, which represents the output of a number channels from, say, seismic instruments. This data is then passed through the frame libraries to construct the frames. Various forms of data will be passed through the libraries, ranging from one column per one file inputs, to various columns in various files, with files describing the data that is to be passed through the libraries. It is likely the construct of the arbitrary data be to a certain specification, likely to be found in the literatures reviewed. The arbitrary data needed will be provided for by the ACIGA group.

Results will be documented, analysed and used as a basis for the applications implementation.

How will a feasibility study be carried out to ensure an accurate cost/benefit analysis?

Basically this is a two step procedure. 1) To test the efficiency of the frame libraries. I.e. how many tasks need to be executed to reach a certain goal? 2) Using the same method, only with a semi constructed prototype of the GUI to obtain a similar measurement.

5.1.3 Requirements Limitations

Two main limitations exist that may fetter the study impartially.

1. Due to frame libraries being used by only a small audience, information concerning its system is limited, although the C code is freely published. This brings the number 2 limitation...
2. Comprehensive knowledge of the C language is required for understanding the published code. C is a third generation language requiring time to grasp its concepts.

5.1.4 Summary of Requirements

1. The ability to create frames from data produced by seismic and other instruments.
   a. This includes batch processing data files. It is insufficient if files must be processed manually one at a time.
2. The ability to read these frame files, and to read the specific contents of them.
3. The ability to manipulate the frame files, e.g. extract data from only certain parts of the frame files, add new data to the frame files, etc.
4. Ensure all frame files coming in or leaving from the ACIGA workstations pass through inspection of integrity so as to adhere to the requirements of foreign detector sites.
5. The ability to do the above through a simple to learn, simple to use graphical user interface.

5.2 *Preliminary Investigation*

A preliminary investigation assisted in selecting the right development environment to produce the graphical interface. The conclusion of this preliminary investigation, as most likely concluded already, is the QT environment. But it is important to briefly discuss how, and more specifically why the preliminary investigation came to this conclusion.

5.2.1 Specific QT investigation

Linux Based

The frame libraries and utilities currently only exist under Unix based operating systems. Due to this necessity, it was left to examine C++ graphical environments under Linux. Linux is a fairly new operating system, particularly in the field of Windows type environments (i.e. X windows, Gnome and KDE), the monopolies on graphical development environments are restricted to Gtk (Gnome Tool Kit) and KDevelop (for developing KDE native applications) and QT. QT stands out has being portable across both operating systems, but has the unique capability of also being portable to windows based systems.
"On Unix systems, QT is the best option; it's portable, fast, and easy to use…if you want to develop not only for Unix and MS Windows, but also for Macintosh, there is no alternative but QT. If you are developing for embedded systems (notably, embedded systems running a version of Linux), you would have a hard time finding any GUI toolkit that comes close to QT in terms of functionaliy"[7]

Furthermore, "*(22. Aug 2002)* - SANTA CLARA, California - Trolltech, the leader in single-source, multiplatform software development, today announced that Qt/Embedded won the "Best Embedded Linux Solution" at last week's LinuxWorld Conference & Expo. This is the second straight year Trolltech's Qt/Embedded has won this award."[8]

[7] Dalheimer, M.K. (2002) Programming with QT 2nd Edition : O'Rielly
[8] Trolltech Technologies. Trolltech Wins "Best Embedded Solution" for Second Straight Year (2002)

In terms of simplicity; "in their background for the decision (for the award), the judges explain, "Qt has by far the more elegant object-oriented library of classes. Qt excels because it is a more abstract class library [...] which minimizes the amount of work the programmer must do in order to write applications.""[9]

5.3 Materials

Preliminary research (above) has gathered that a reliable and efficient tool to interface to the frame library is the QT C++ GUI application framework X11 edition (i.e. Linux edition), see section above.

Hence, it is critical to understand and analyse the two components essential for the GUI to be investigated further, in particular, concentrating on the most relevant elements necessary to fulfil the derived requirements. These are the frame libraries and associated utilities, and offcourse, the QT development environment. These will now be comprehensively analysed.

5.3.1 Frame Libraries Analysis

Answering: How much information on the frame libraries is available and is it freely accessible?

The information derived from the requirements analysis leads to a contained number of frame utilities needed to conclude the development of the graphical interface.

The list of frame manipulating utilities required is listed below:

- Ascii2frame
- Table2frame
- FrameDataDump
- FrCheck
- FrCopy

[9] Trolltech Technologies. (1999) Qt 2.0 Wins LinuxWorld Award

- FrDump
- FrSplit

Each of the listed utilities has specific options in which to run with. For a list and a full description of these options, please refer to appendix 9.2.

It is very important to differentiate between frame libraries and frame utilities to limit confusion of the two. Frame utilities are listed above. Frame utilities basically are C applications which interface to C libraries, and allow the creation, reading, checking, and splitting of frame files. The GUI will interface to the frame utilities, which in turn, interface to the frame libraries. Note, as a pre-requisite, the frame utilities will not run without the frame libraries, and the GUI will not operate without the frame utilities.

Discussing the Frame Library (Fr) version 6 release 02 (October 22, 2002).

The base libraries needed to run the frame manipulation applications/utilities are adequately called, the Frame Library. "The Frame Library is a software dedicated to the frame manipulation including file input/output." The basic C structures which interfaces to upper-level programs e.g. FrDump (which will be discussed further below) are listed below.

- Library control
- Frame Handling
- Input File: FrFileI
- Output File: FrFileO
- File checksum
- Error Handling
- FrAdcData
- FrDetector
- FrHistory
- FrMsg

- FrProcData
- FrSerData
- FrSimData
- FrSimEvent
- FrStatData
- FrSummary
- FrTable
- FrEvent
- FrVect

Whilst specific detail in how these libraries operate is beyond the scope of the research, it is important to briefly discuss how the following utilities, which are needed for the GUI to interface to, may be affected by the libraries. The utilities incorporate, or make reference to, these libraries in their code. The utilities cover the range of tasks requested to be handled for the ACIGA research group (see section 5.1.4 for a summary of requirements). The following utilities cover such tasks. Their practical use is covered in section 6.2, where examples can be found.

Ascii2frame

"ascii2frame creates frames containing several channels at different sampling rates, starting from several series of ASCII files."[10]

The Ascii2Frame utility inputs a file which contains a single column of data, and outputs a single frame file. The frame file outputted named similar to P-715231934.F where the number represents the time at which the file was created. The time is given in Unix local time.

[10] Vicere, A. (2002) Frame builder emulation (Fbe) library v1r2

Table2frame

"table2frame converts a single column-formatted file into frames with several channels all at the same sampling rate."[11]

By default, table2frame stores the data in SimData structures in the frame file. Looking at the options (see section 9.2) which come with table2frame, we one such option being –type.

The type can be adc, proc, sim. Default sim. Please use

- adc for raw data (adc counts)
- proc for processed data
- sim for simulated data.

This has implications when reading the frame file through FrameDataDump. SimData structure is not read by FrameDataDump, for the only data structures which are read are adc (raw data). Thus if we produce a frame file without using the option –type adc, we would get an error when reading the frame file through FrameDataDump.

FrameDataDump

This utility extracts data stored in a frame file, and allows the options of outputting to a binary file, or to the screen. It is limited that it allows the output of one channel at a time. This is where a GUI to this utility is practically useful, in that the application may signal output of data from a single channel, multiple times automatically.

FrCheck

"This program checks that the frame file could be read successfully. The file checksum are also checked if they are available. In case of success, this program returns a positive value, the number of frames in the file. It returns a negative value in case of error."[12]

[11] VIR-MAN-LAP-5400-103, (2002) Frame Library (Fr) Users Manual

This tool is particularly important to use when exchanging frame files between detector sites. Checking

FrCopy

"This program reads frames from one or more input files, merge them if requested and write them to one or more output file, using or not data compression. See the help function bellow for program use."[13]

5.3.2 QT Interest Group

As suggested in the Literature Review section, limited literature is available which helps users of QT in technical aspects of its use. There does exist however an online QT-Interest group which consists of QT users, whom discuss anything related to QT, from examples of code, to more theoretical know-how. Being part of this group has been particularly useful, since many practical problems may be asked, and a response from experienced users and staff is likely to come with a solution to most problems.
Previous questions are archived which form a large searchable knowledge base, which is also an extremely helpful material which assists with the research and development.

5.3.3 QT Libraries Analysis :

This section explores the QT libraries. The combinations of these libraries really make up the whole of the QT development environment. Let us take the QButton library for example. The QButton is known as a "widget". A widget is a re-usable software component which can have generic attributes given to it, such as, size, colour, label, etc. In the example of the QButton widget, it is basically a standard button which is so prominent in all GUIs.

[12] VIR-MAN-LAP-5400-103, (2002) Frame Library (Fr) Users Manual
[13] VIR-MAN-LAP-5400-103, (2002) Frame Library (Fr) Users Manual

Figure 5.3.3.1, QButton widget.

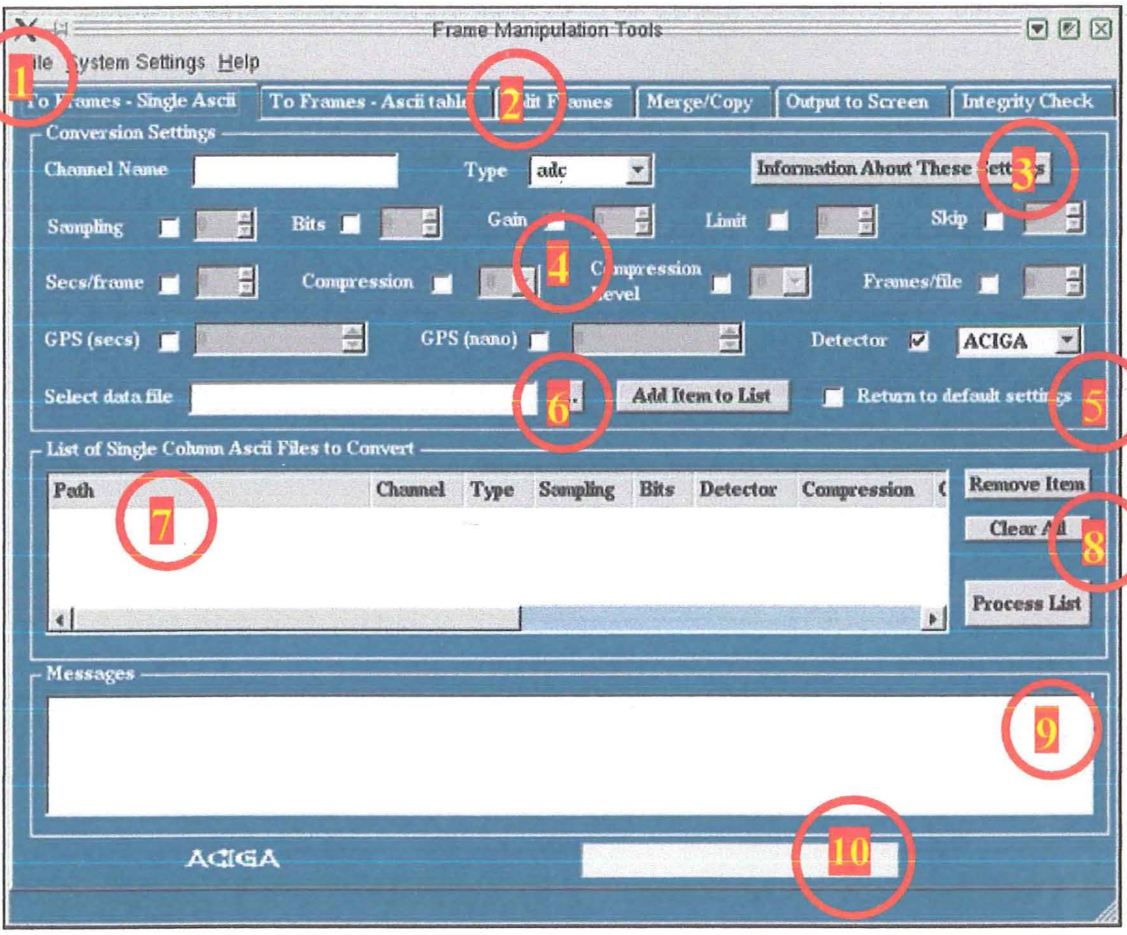
The QT libraries are vast, with 10's of widgets available for all sorts of tasks. Our main concern is with widgets which are relative, and would be used in, the interface's development. After investigation of the widgets available, the following are deemed to be most appropriate to use in the interface:

QPopupMenu	Popup menu used to link sections of the application, e.g. either then the File or Edit menus, and System Settings menu may be added to give the users the ability to tailor the application.
QCheckBox	The frame libraries come with options that may be attributed to a file which is to be run through the libraries, e.g. the ascii2frame file has the –d option, which adds a detector name to the frame file. A check box may be checked if they wish to include this option.
QComboBox	Used for options with a limited and confined attributes. For example the –type option is restricted to three elements, adc, sim, or proc. It is ill advisable to allow the user to enter their choice, due to naturally occurring mistakes. A combo box ceases mistakes from occurring.
QLineEdit	Line to enter file path.
QListView	List of files that have been added, with their selected options.
QPushButton	The push button shall be clicked on to commit such tasks as, Add Item to list, Process list, etc.
QSpinBox	Used for options restricted to numbers only.
QTabWidget	The interfaces to the frame applications, e.g. ascii2frame, table2frame etc will be split by tabs. If a user wishes to interface to the ascii2frame library, they select the appropriate tab.

Table 5.3.1 Main widgets used for the graphical interface.

5.3.4 Interface Design Prototype

The following interface was created using QT native C++ code. For the purposes of abiding by conventional prototype reasoning of use, and not overdoing the requirements, it is deemed sufficient that an interface be created which only portrays the connection to the ascii2frame utility.



Type adc

Detector ✓ ACIGA

Figure 5.3.4.1 Frame Manipulation Tool interface prototype.

1. Menu 2. Tab separators 3. Information on the settings 4. Settings to set before creating frame file 5. Choice to return settings back to default after adding a file to the list.	7. List of files with descriptions of settings for each file, ready to be processed. 8. Remove button to remove selected item from list, Clear All button to remove all items from list, and a process list button to create frame

6. Selected file path.	files. 9. Feedback messages returned whilst compiling the list. 10. Progress bar shows how far into the batch process the application is in.

The code for this interface can be found in appendix section 9.3.

One may notice with the naming conventions of the widgets is that they begin with Q. Obviously this is somehow related to QT. The Q is present to make a distinction between QT specific libraries, which has its own set of rules and modified laws, with standard C++ ANSI conventions.

6.0 Methodology and Results

The methodological approach taken is simple and straight forward.

The method for creating frame files is through using arbitrary data which is gathered from seismic instruments, and plain conjured data. This data is passed through the utilities by both command line arguments and automated QT native C++ code.

The created frame files are then used to test the utilities which read, copy, check and split frames.

The outcome of these tasks is discussed in section 7.0 (Findings).

6.1 Arbitrary data

Data used to pass through the frame libraries and utilities are very specific in the way it must be structured in order for the outputted frame file maintains integrity.

Ascii2frame requires data to be "separated by spaces and/or tabs and/or newlines"[14] Therefore the two sample data used for the Ascii2frame utility are sample1.data and sample2.data.

Table 6.1.1 Ascii_sample1.data contents (note, this data is not in columns, although it appears so for obvious reasons, it is in fact one long line of data).

0.134386001	0.791153962	0.360409309	0.392439577	0.314344885
0.017116691	0.654190591	0.186685965	0.159960835	0.314728504
0.002161153	0.215129909	0.550453498	0.05624062	0.807946738
0.003319073	0.227756032	0.221669527	0.110173133	0.081753268
0.658575545	0.106246538	0.197627016	0.417980481	0.59303601
0.848221333	0.357054973	0.015491877	0.981957455	0.638099036
0.373528844	0.146437451	0.377246218	0.58519152	0.343893072
0.274018182	0.881383025	0.347487023	0.086053982	0.972393313
0.801217907	0.647346491	0.184241525	0.49098594	0.330341071

[14] California Institute of Technology. (2002) Laser Interferometer Gravitational Wave Observatory

```
0.719185319 0.348997847 0.37675773  0.744136588 0.00422898
0.847754088 0.679425357 0.310810314 0.156002391 0.297214609
0.127185204 0.80245554  0.866570729 0.59593272  0.249606415
0.482744963 0.741296346 0.856891352 0.890097298 0.048943974
0.526610082 0.397751276 0.169960943 0.637812711 0.043419864
0.102925577 0.364615782 0.368817156 0.326052081 0.916720774
0.982863801 0.242212762 0.842789012 0.284964002
```

Table 6.1.2 ascii_sample2.data contents

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
```

Table2frame on the other hand, requires data be contained in 2 separate files. First file containing various columns of data, and the 2nd file describing each column.

Table 6.1.3 table_sample1.data

```
0.1     0.777536777 0.657849614 0.478500865
0.2     0.079988414 0.203283426 0.548240217
0.3     0.74797563  0.799714078 0.19767202
0.4     0.991196075 0.42476208  0.730983234
0.5     0.74430289  0.146442667 0.337391269
0.6     0.094966205 0.183477764 0.332463707
0.7     0.815892846 0.891759714 0.272489679
0.8     0.481988043 0.20666739  0.124026185
0.9     0.851264704 0.268098497 0.766002719
1.0     0.536466749 0.67763535  0.338107593
1.1     0.271390402 0.923680512 0.039316442
1.2     0.949094803 0.961074468 0.757950106
1.3     0.907722001 0.962707438 0.260314288
1.4     0.696741309 0.926560572 0.686696771
1.5     0.368138215 0.304217008 0.900364054
1.6     0.95551948  0.092362579 0.103929237
1.7     0.736558822 0.991027496 0.57508325
1.8     0.568325551 0.746846523 0.95238405
```

table_description1.desc

```
time ch1 ch2 ch3 ch4
```

6.2 Dissecting Tasks

The following sub sections will portray two methodologies:

1. Typical methods various frames are created and manipulated not using the GUI. It is important to understand the workings of these utilities in order to allow the interfaced application perform the same tasks but in an automated manner. Note, an explanation of the options which are used in the command line examples can be found in appendix 9.2.

2. The actual automated tasks being performed by the GUI using native QT C++ code, including error handling. The example we use here is the interface to the ascii2frame utility.

All code is located in appendix section 9.4.

6.2.1 Writing Frames

Creating frame files using the Ascii2frame utility

Manually-Command Line arguments

Sample one:
```
ascii2frame –channel ch1 –sampling 1024 –I ascii_sample1.data –type adc
```

Output one:
```
P-723158926.F
```

Sample two:
```
ascii2frame –channel ch1 –sampling 1024 –I ascii_sample2.data –type adc
```

Output two:
```
P-723158930.F
```

Creating frame files using the Table2frame utility

Manually-Command Line arguments

Sample one:

```
table2frame -i table_sample1.data -description table_sample1.desc -type adc -type adc –
type adc
```

Output one:

```
P-723158945.F
```

6.2.2 Reading Frames

Reading Frame files through FrameDataDump utility

Manually-Command Line arguments

Sample one:

```
FrameDataDump –IP-723158926.F –Cch1 –d -a
```

Output one:

```
0.134386 0.791154 0.360409 0.392440 0.314345 0.017117 0.654191 0.186686 0.159961
0.314728 0.002161 0.215130 0.550453 0.056241 0.807947 0.003319 0.227756 0.221670
0.110173 0.081753 0.658576 0.106247 0.197627 0.417980 0.593036 0.848221 0.357055
0.015492 0.981957 0.638099 0.373529 0.146437 0.377246 0.585192 0.343893 0.274018
0.881383 0.347487 0.086054 0.972393 0.801218 0.647346 0.184242 0.490986 0.330341
0.719185 0.348998 0.376758 0.744137 0.004229 0.847754 0.679425 0.310810 0.156002
0.297215 0.127185 0.802456 0.866571 0.595933 0.249606 0.482745 0.741296 0.856891
0.890097 0.048944 0.526610 0.397751 0.169961 0.637813 0.043420 0.102926 0.364616
0.368817 0.326052 0.916721 0.982864 0.242213 0.842789 0.284964

1 files opened
79 total samples (1.000 s) read and 79 (1.000 s) written to standard out
The sampling rate is 100.000000 samples/sec
The data output was continuous.
```

Sample two:

```
FrameDataDump –IP-723158930.F –Cch1 –d -a
```

Output two:

```
0.000000 1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000 8.000000
9.000000 10.000000 11.000000 12.000000 13.000000 14.000000 15.000000 16.000000
17.000000 18.000000 19.000000 20.000000 21.000000 22.000000 23.000000 24.000000
25.000000 26.000000 27.000000 28.000000 29.000000 30.000000 31.000000 32.000000
```

```
33.000000 34.000000 35.000000 36.000000 37.000000 38.000000 39.000000 40.000000
41.000000 42.000000 43.000000 44.000000 45.000000 46.000000 47.000000 48.000000
49.000000 50.000000

1 files opened
50 total samples (1.000 s) read and 50 (1.000 s) written to standard out
The sampling rate is 100.000000 samples/sec
The data output was continuous.
```

Sample three:
```
FrameDataDump –IP-723158945.F –Cch1 –d -a
```

Output three:
```
0.777537
0.079988
0.747976
0.991196
0.744303
0.094966
0.815893
0.481988
0.851265
0.536467

1 files opened
10 total samples (1.000 s) read and 10 (1.000 s) written to standard out
The sampling rate is 10.000000 samples/sec
The data output was continuous.
```

Sample four:
```
FrameDataDump –IP-723158945.F –Cch2 –d -a
```

Output four:
```
0.657850
0.203283
0.799714
0.424762
0.146443
0.183478
0.891760
0.206667
0.268099
0.677635
```

```
1 files opened
10 total samples (1.000 s) read and 10 (1.000 s) written to standard out
The sampling rate is 10.000000 samples/sec
The data output was continuous.
```

Sample five:

```
FrameDataDump –IP-723158945.F –Cch3 –d -a
```

Output five:

```
0.478501
0.548240
0.197672
0.730983
0.337391
0.332464
0.272490
0.124026
0.766003
0.338108

1 files opened
10 total samples (1.000 s) read and 10 (1.000 s) written to standard out
The sampling rate is 10.000000 samples/sec
The data output was continuous.
```

6.2.3 Checking frame file integrity

Checking frame file integrity through FrCheck utility

Manually-Command Line arguments

Sample one:

```
FrCheck –i IP-723158926.F
```

Output one:

```
Test file P-723158926.F
   Scanning the file Please wait....
      1 frames read
  1 frame(s) in file
 No read error
 Checksum OK (2dc89267)
```

Sample two:

```
FrCheck –i IP-723158945.F
```

Output two:

```
Test file P-723158945.F
   Scanning the file Please wait....
      1 frames read
  1 frame(s) in file
No read error
Checksum OK (ef0992b9)
```

6.2.4 Copying/Merging frames

Copying and Merging frames through FrCopy utility

Manually-Command Line arguments

Sample one:

```
FrCopy -i P-723158945.F –i P-723159876.F –f 723158945
```

Output one:

```
P-723159987.F
```

6.2.5 Splitting

Splitting multiple frame files through the FrSplit utility

Manually-Command Line arguments

Sample one:

```
FrSplit –o /ligoapps/ligotools/testing –i P-723159987.F –n 1
```

Output one:

```
P-723158945.F and P-723159876.F in the directory /ligoapps/ligotools/testing
```

6.3 Automating Tasks

Three basic criteria are prevalent and need interfacing from QT code, to the frame libraries, in order to allow automation of processing.

1. The ability to select and set the options which come with each library.
2. The ability to select multiple files to process at once.
3. The ability to run the frame libraries processes from within the GUI, i.e. to start and end the process.

The selection of options can take place through checkboxes. E.g.

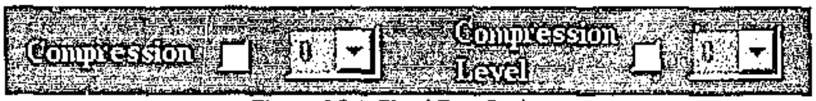


Figure 6.3.1 CheckBox Options.

if one wishes to use the –compress and –gzip options, the simply check the appropriate QCheckBox and the relative QComboBox will appear. When processing, the following C++ code will be executed to establish whether these options are to be attributed to the file being converted or not:

```
if ( CheckBox9->isChecked() == TRUE)
        {compressionLevel1 = ComboBox1 -> currentText();}
else
        {compressionLevel1 =  "-";}
```

To mass process files, one adds the files with the options to the QListItem.

```
QListViewItem *item = new QListViewItem( ListView1);

  item->setText(0,path1);//path
  item->setText(1,channel1);//channel
  item->setText(2,type1);//type
  item->setText(3,sampling1);//sampling
  item->setText(4,bits1);//bits
```

```
   item->setText(5,detector1);//detector
   item->setText(6,compression1);//compression
   item->setText(7,compressionLevel1);//compression rate
   item->setText(8,secsFrame1);//secs/frame
   item->setText(9,limit1);//limit
   item->setText(10,framesFile1);//frame file
   item->setText(11,gpsSec1);//gps(secs)
   item->setText(12,gpsNano1);//gps(nano)
   item->setText(13,skip1);//skips
   item->setText(14,gain1);//gain
```

Finally, the actual processing takes place. Notice QTs unique ability to start a command line process from within the actual code, parsing the appropriate options to the new process, and receiving any feedback.

```
proc = new QProcess( this );

proc->addArgument( "/ligoapps/ligotools/bin/ascii2frame" ); //path to actual library.
proc->addArgument( "-channel" ); // channel name
proc->addArgument( channel1 ); // String variable holding the name of the channel
proc->addArgument( "-i" ); // path of file option
proc->addArgument( path1 ); // String variable holding path to file
proc->start(); //trigger to execute the above commands and initialize process.
```

A simple Repeat...while number_of_items > 0 loops is sufficient to automate the process.

7.0 Findings

Answering: What will be the improved speed and effort of execution of conversion of frames files?

7.1 Efficiency Metrics

A simple efficiency metric calculates the time it takes from the beginning to the end of a task to execute. This includes human interaction.

Four scenarios take place:

Manual command line system	Automated GUI system
1. ascii2frame executed with 5 various options set. Repeated once. Path to file /usr/local/data/anonemeter/2_1_2002/	2. ascii2frame executed with 5 various options set. Repeated once. Path to file /usr/local/data/anonemeter/2_1_2002/
3. ascii2frame executed with 5 various options set. Repeated 20 times. Path to file /usr/local/data/anonemeter/2_1_2002	4. ascii2frame executed with 5 various options set. Repeated 20 times. Path to file /usr/local/data/anonemeter/2_1_2002

Table 7.1.1 Efficiency metrics scenarios.

Scenario one:

"ascii2frame –channel ch1 –i /usr/local/data/anonmenter/2_1_2003 –detector aciga –seconds_per_frame 1 –sampling 1024" is typed into the command line.

Approximately 35 seconds.

Scenario two:

1. Options set.
2. File selected through popup dialogue box directory navigator.
3. File added to list.
4. List executed.

Approximately 15 seconds.

Scenario three:
"ascii2frame –channel ch1 –i /usr/local/data/anonmenter/2_1_2003 –detector aciga –seconds_per_frame 1 –sampling 1024" is typed 20 times into the command line. *Approximately 14 minutes.*

Scenario four:
1. Options set.
2. File selected through popup dialogue box directory navigator.
3. File added to list.
4. List executed.
5. Repeat the above 20 times.

Approximately 6 minutes.

The completed GUI interface prototype was exposed to the UWA ACIGA project group for assessment.

7.2 Ease of Use and GUI Effectiveness

The above timings were taken from users with reasonable computer skills. The results were quite obvious. No formal questionnaires or interviews took place for feedback concerning the use of the GUI, only informal discussions.

For users with limited computer technical know-how, using the manual method was ambiguous. A user interface clearly simplified the process of creating frame files. For those with greater computer experience, undoubtedly the interface had sped up a lot of the conversion process.

The most difficult task was using the QT development environment. Its own native concepts meant virtually a new language had to be learnt. Limited literature and support documentation also cause a steep learning curve to climb. However, once these basic schemes were understood, and due to assistance from the qt-interest group, development became straight forward.

7.3 Possible Future Directions

QT is a cross-platform application development toolkit, and is the only graphical interface toolkit that has the capability of migrating code between various operating systems and compiling. Linux has long had the reputation of being the "geeks" choice of computer, signifying the need for more computing skills to use. It is of this advantage, that the application may be migrated to a windows machine and compiled.
There are limitations to this however. The QT development environment under Linux is freeware and freely distributed. However under Microsoft Windows, the environment costs hundreds of dollars to purchase.

GUI development under Linux environment is by far less available then that under windows. As the growth trend of Linux continues, it is likely so will various tools to assist in GUI developments.

Constant updates will also pay a key in the future roll of the GUI. The gravitational wave detection project is dynamic. No doubt the frame libraries will evolve, meaning the GUI will have to adjust with these changes.

7.4 Research Limitations

Two major factors limited the research.

1. Lack of literature concerning QT: compared to other development applications such as Visual Basic, the documentation and knowledge base of QT is extremely small. Programming books are focal in assisting a programmer develop applications.
2. The very nature of the research: in essence, the research is not really considered pure research. Research involves a lot of theoretical investigation. This project however involved a lot of practicality, i.e. constructing a graphical interface. This limits the research in that the researcher is limited to a set capacity of matter to explore, and consequently elaborate upon. This is mainly due to the fact that the research is at honours level, and confined to a short strict time barrier.

8.0 Conclusions

Frame libraries

Judging from the use of the frame libraries and utilities, it is obvious the task of creating, reading and manipulating frame files can be time consuming, as well as confusing to those who are not accustomed to the Unix "way of doing things". This is particularly true if it is required to process the immense quantity of data that will eventually be produced by the ACIGA detection facilities.

Graphical User Interface

Automated graphical user interface is the best option to assist in the processing of the potentially vast amounts of data. Whilst maintaining all the available options of the library, the GUI helps batch process multiple files from multiple locations, something not possible to do through one task using the frame utilities and libraries, severely cutting down dispensation time.

Use of QT

QT proved to be an extremely simple to use graphical development environment. The C++ libraries consist of ready to use objects which allowed the connection to the frame libraries and the ability to select options, just as one selects option through the command line.

QT provides the added ability of future portability, in that the code can be recompiled under windows, or Macintosh systems; something which no other development tool allows.

9.0 Appendices

9.1 Frame Format

The LIGO/VIRGO Data Frame Format for Interferometric gravitational wave detectors (IGWD) is a collaborative effort which has evolved out of a frame format design originated within the VIRGO Project. This specification has evolved out of the recognition for the need of a standard definition of this frame format which can be used by individual (international) projects wishing to adhere to a common representation of data produced by IGWD. It is hoped that by using a standard design for data, future collaborative analyses of data taken by different projects can be promoted more easily.

The predominant type of data stored in frames is time series data of arbitrary duration. It is possible, however, to encapsulate in frame structures other types of data, e.g., spectra, lists, vectors or arrays, etc. However, the primary purpose of this specification is to address how (raw) data are written into frame structures.

It is the intent of the Projects collaborating internationally on this frame definition and standardization to promote a continued evolution of the standard through formal configuration control, scheduled updates and releases, and code maintenance. A Consortium or Working Group with representatives from each of the participating projects will be formed and will have formal control over the contents of this specification as it evolves.[15]

[15] VIRGO-SPE-LAP-5400-102, (2002) Specification of a Common Data Frame Format for Interferometric Gravitational Wave Detectors (IGWD)

Figure 9.1.1 Frame file format makeup

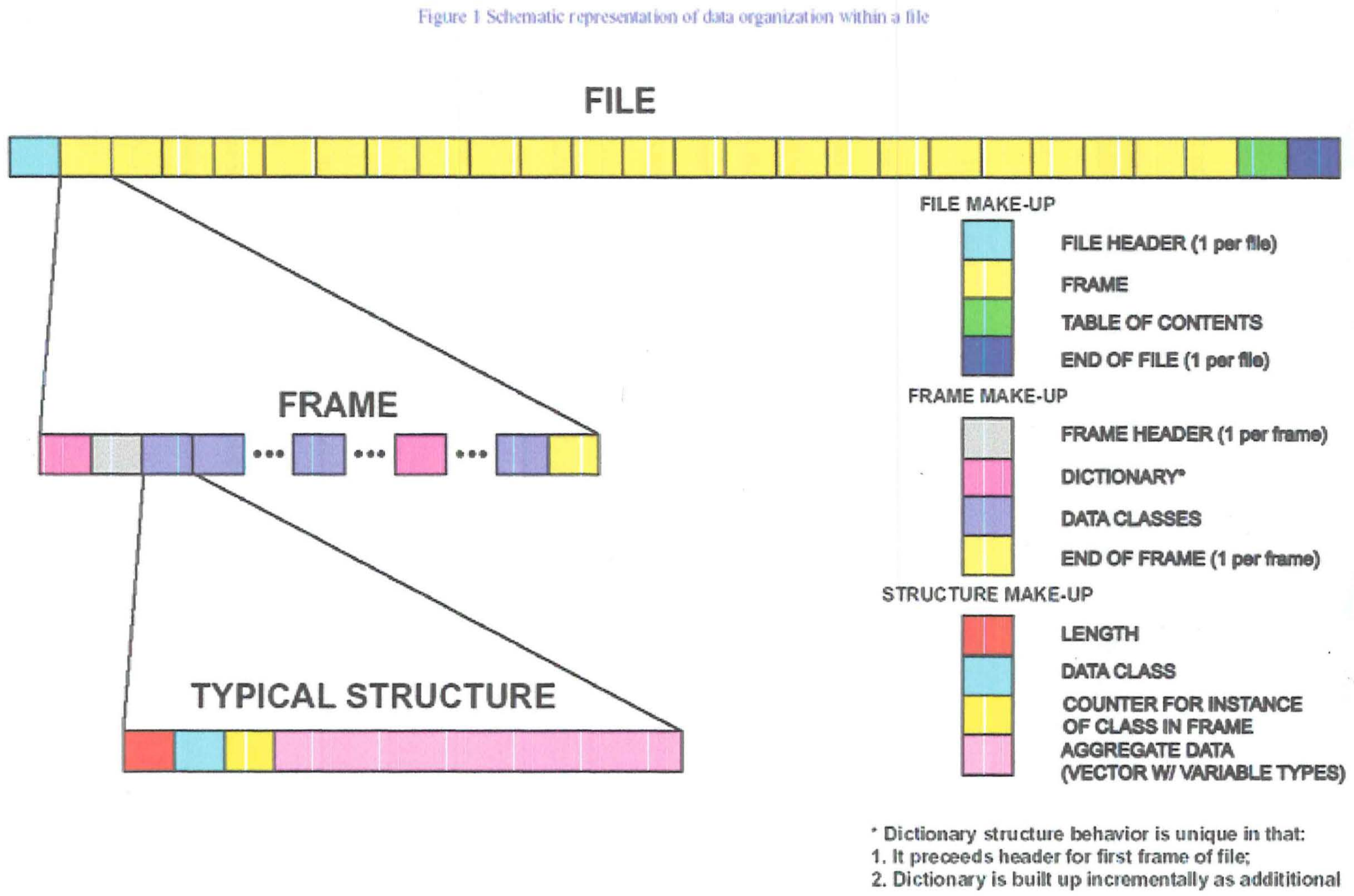


Figure 1 Schematic representation of data organization within a file
FILE
FRAME
TYPICAL STRUCTURE
FILE MAKE-UP
FILE HEADER (1 per file)
FRAME
TABLE OF CONTENTS
END OF FILE (1 per file)
FRAME MAKE-UP
FRAME HEADER (1 per frame)
DICTIONARY*
DATA CLASSES
END OF FRAME (1 per frame)
STRUCTURE MAKE-UP
LENGTH
DATA CLASS
COUNTER FOR INSTANCE OF CLASS IN FRAME
AGGREGATE DATA (VECTOR W/ VARIABLE TYPES)
* Dictionary structure behavior is unique in that:
1. It preceeds header for first frame of file;
2. Dictionary is built up incrementally as addititional

9.2 Frame Utilities

The following frame utilities consist of options needed to be included when the utilities are executed under a command line (or in Linux terms, a shell). For examples of command line arguments, please refer to section 6.2.

9.2.1 Ascii2frame

http://www.ldas-sw.ligo.caltech.edu/ligotools/Fbe/html/ascii2frame_8c-example.html

The available options are:

- -detector <name> : give this "name" to the detector
- -channel <name> : create a channel with this name
- -sampling <float> : set the sampling for this channel
- -gain <float> : set the gain for the current channel
- -bits <int> : bits to use to encode this channel
- -type <adc|sim|proc> : set the type of channel
- -i <filename> : add filename to the inputs for this channel
- -skip <int> : skip these many chars at the beginning of the current file.
- -limit <int> : limit on the number of items to read from the current file (e.g.: -limit 5 means read 5 numbers.)
- -o <filename> : output base filename>
- -compress <int> : compression type
- -gzip <int> : gzip compression level
- -debug <int> : debug level [default 0]
- -GTimeS <int> : starting frame time in GPS seconds
- -GTimeN <int> : starting frame time, GPS nanoseconds
- -seconds_per_frame <int> : seconds for each frame
- -frames_per_file <int> : how many frames per file

Remarks:

Compression can be

- 0 no compression
- 1 gzip
- 3 differentiation + gzip
- 5 differentiation + zero_suppression, only for short
- 6 differentiation + zero_suppression for short. gzip for others. The default is 6

Remarks:

The gzip level ranges from 0 (no compression) to 9 (maximum, slow). Default 1

Remarks:

The type can be adc, proc, sim. Default sim. Please use

- adc for raw data (adc counts)
- proc for processed data
- sim for simulated data.

Remarks:

The allowed number of bits depend on the type:

- proc or sim:
    - if bits < -32, double format is used
    - if bits >= -32 float format is used
- adc:
    - if bits > 16, long int are used
    - if bits > 8, short int
    - if bits > 0, char
    - if bits < -32, double
    - default float

Remarks:

The recognized names for the detectors (case unsensitive) are:

- TAMA_300 [Tokyo, JPN]
- VIRGO [Cascina, ITA]
- GEO_600 [Garching, GER]
- LIGO_HANFORD_2K [Hanford, WA, USA]
- LIGO_HANFORD_4K [Hanford, WA, USA]
- LIGO_LIVINGSTON [Livingston, LA, USA]
- PROTOTYPE [Caltech, 40m, Pasadena Ca]
- ALLEGRO [Baton Rouge, LA, USA]
- AURIGA [Padova, IT]
- EXPLORER [CERN, CH && FR]
- NIOBE [Perth, AU]
- NAUTILUS [Frascati, IT]

Note: The above remarks also apply to the following utility (i.e. table2frame)

9.2.2 Table2frame

http://www.ldas-sw.ligo.caltech.edu/ligotools/Fbe/html/table2frame_8c-example.html

The switches which can be used are the following:

- -detector <name> : give this "name" to the detector
  - -i <file>: read data from "file", should be in tabular format
  - -skip <int>: how many chars to skip at the beginning of the input file
  - -comment <char>: which comment to assume for the input file
  - -description <file>: read from "file" the channel names
  - -sampling <float>: sampling frequency
  - -channel <name>: set the current channel
  - -type <adc|proc|sim>: the kind of data to store in current channel

- -gain <float>: the gains to apply to the current channel
- -bits <int>: the bits to encode the current channel
- -GTimeS <int>: set starting time in seconds
- -GTimeN <int>: set nanosecond portion of the starting time
- -seconds_per_frame <float>: seconds for each frame
- -frames_per_file <int>: how many frames per file
- -o <string>: output base file name, shall be completed with other informations, including the GPS time.
- -compress <int>: compression type
- -gzip <int>: gzip level
- -debug <int>: debug level

9.2.3 Frame Data Dump

- -I<path> --The path to the frame directory as well as any file search parameters or a single file name. If search parameters are used the option must be in single quotes, ie -I'path/*'
- -O<path> --The path to the file in which to output the data
- -C<channel> --The channel to extract data from
- -N<samples> --The number of samples to extract, -1 for all available points
- -T<seconds> --The number of seconds of data to extract. Use in lieu of -N
- -Sn<samples> --The number of samples to skip from the frame files before output
- -St<seconds> --The number of seconds of data to skip. Use in lieu of -Sn
- -Sf<files> --The number of complete frame files to skip before output
- -D<#> [file] --Output frame library debug info to file; if file is stdout, output
  - will be to the screen; # = debug level = 1(min) to 4(max) (note

- the space between the number and the file name)
- -z --Place zeros for any time gaps between frames
- -d --Display sample values to the screen in lieu of writing a file
- -s --Force short int's to be written to the binary file
- -f --Force float's to be written to the binary file
- -a --Supress all output to standard out except that of the time series This is useful to output data to an ASCII file.

9.2.4 Frame Check (FrCheck)

http://wwwlapp.in2p3.fr/virgo/FrameL/FrDoc.html

- -i <input file> Only one file should be used
- -d <debug level> (default 1). 0 will suppress all info and error messages.
- -t to scan the file using only the TOC
- -s to scan the file using only sequential read(TOC not used)
- -f GPS time of the first frame to scan (default=0) (only used when doing the random access)
- -l GPS time of the last frame to scan (default: 999999999.) (Only used when doing the random access).
- -h to get the help

9.2.5 Frame Copy (FrCopy)

http://wwwlapp.in2p3.fr/virgo/FrameL/FrDoc.html

- -i <input file(s)> If more than one files is given after the keyword -i they will be read in sequence. If several input stream are defined (several -i followed by name(s)), then the frame content will be merged

- -o <output file>
- -f <first frame: (run # frame #) or (GPS time)> Example: -f 0 20 will start with run 0 frame 20. -f 6544444 will start at GPS time = 6544444. If this option is used, the Table Of Content is mandatory and all frames will be read by increasing time.
- -l <last frame: (run # frame #) or (GPS time)>
- -c <compression type> Compression types are -1 (same compression as input file), 0 (no compression), 1 (gzip), 3 (differenciation+gzip), 5 (gzip for float+zero suppress for int), 6 (zero suppress for int). The default is 6.
- -a <list of input adc channels>.When this option is used, random access are performed to read only the requested adc channels. Only the Frame header is returned in addition to the adc data. Additional information like the history records is not returned.
- -t <list of tag channels> Tag is a channel list with wild cards like: ADC122 ADC5* If a name start by - it is interpreted as an anti-tag
- -r <new frame length in second> The reshape option works only with one output file. It assumes that the length of the input frame is a integer number of second. The starting GPS time of the output frame will be a multiple of the frame length. The requested length should be larger than the input frame length.
- -decimate <number of sample to average> The decimation is done on all selected channel by doing a simple data averaging.
- -d <debug level> (from 0 to 5)
- -noTOCts to not write TOC for time series
- -noChkSum to not put checksum in the output file.
- -h (to get the help)

9.2.6 Frame Dump (FrDump)

http://wwwlapp.in2p3.fr/virgo/FrameL/FrDoc.html

- -i <input file(s)> If more than one files is given after the keyword -i they will be read in sequence. If several input stream are defined (several -i followed by name(s)), then the frame content will be merged

- -f <first frame: (run # frame #) or (GPS time)> Example: -f 0 20 will start with run 0 frame 20. -f 6544444 will start at GPS time = 6544444

- -l <last frame: (run # frame #) or (GPS time)>

  -t <list of tag channels> Tag is a ch[illegible]d list with wild cards like: ADC122 ADC5* If a name start by - it is interpreted as an anti-tag

- -d <debug level> (from 0 to 5)

- -h (to get this help)

- If one of the next option is there, we do only a partial frame dump

```
-adc   to dump only the FrAdcData information
-sms   to dump only the FrSerData information
-proc  to dump only the FrProcData information
-sim   to dump only the FrSimData information
-sum   to dump only the FrSummary information
 -stat to dump only the static information
-raw   to dump only the raw data information
-event to dump only the FrEvent and FrSimEvent
```

9.2.7 Frame Split (FrSplit)

Syntax:

```
FrSplit  [-o <dir1 dir2 dir3 ...>]
         [-n <number of files per dir>]
         -i <file1 file2 file3 ...>
         [-c <compression type>]
```

Remarks:

- -o : The argument is a list of output directories. If this option is not given, then all the output goes into the pwd. If only one output directory name is given and a positive number of files per dir is specified, then a numeric index is appended to the directory name, starting with 0.

  For example: FrSplit -o test -n 10 -i H-input.F
  will output the first 10 frames in H-input.F into test.0, the next 10 into test.1, the next 10 into test.2, and so on.

- -n : The argument is the number of files to put in each directory. If this option is not given, then all the output goes into the first dir given after -o.

- -i : The argument is a list of input frame files. You can use a wildcard, which will be expanded by the shell.
  For example: FrSplit -o test -i H*.n100

- -c : 0 = no compression, 1 = gzip, 3 = differentiate + gzip,
  5 = zero suppress for int, 6 = gzip for float + zero suppress for int
  -1 = same as input (default)

9.3 - QT Interface Code

Three files exist. "Main.cpp" is the binding file which connects the declarations header file "MainWindow.h" to the attributes and application file "MainWindow.cpp." These first three files offer no functionality "MainWindow.ui.h" is the extra code which is entered by the programmer.

9.3.1 Main.cpp

```
/* Muhammed Aoun */
# include <qapplication.h>
# include "MainWindow.h"

int main( int argc, char *argv[] )
{
QApplication app ( argc, argv );

MainWindow MainForm;
app.setMainWidget( &MainForm );
MainForm.show();

return app.exec();
}
```

9.3.2 MainWindow.h

```
/* Muhammed Aoun */
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <qvariant.h>
#include <qmainwindow.h>
class QVBoxLayout;
class QHBoxLayout;
class QGridLayout;
class QAction;
class QActionGroup;
class QToolBar;
class QPopupMenu;
class QCheckBox;
class QComboBox;
class QGroupBox;
class QLabel;
class QLineEdit;
class QListView;
class QListViewItem;
class QProgressBar;
class QPushButton;
class QSpinBox;
class QTabWidget;
class QTextEdit;
class QWidget;

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow( QWidget* parent = 0, const char* name = 0, WFlags fl = WType_TopLevel );
    ~MainWindow();

    QTabWidget* TabWidget5;
    QWidget* tab;
    QProgressBar* ProgressBar1;
    QLabel* TextLabel1_2;
    QGroupBox* GroupBox9;
    QTextEdit* TextEdit2;
    QGroupBox* GroupBox4;
    QPushButton* PushButton8;
    QPushButton* PushButton11;
    QPushButton* PushButton9;
    QListView* ListView1;
    QGroupBox* GroupBox1;
    QLabel* TextLabel4;
    QLabel* TextLabel5;
    QLabel* TextLabel11;
    QLabel* TextLabel15;
    QPushButton* PushButton16;
    QLineEdit* LineEdit2_2;
    QLabel* TextLabel1;
    QPushButton* PushButton12;
    QCheckBox* CheckBox1;
    QLabel* TextLabel6;
    QCheckBox* CheckBox10;
    QLabel* TextLabel12;
    QCheckBox* CheckBox9;
    QLabel* TextLabel7;
```

```
QCheckBox* CheckBox7;
QCheckBox* CheckBox5;
QLabel* TextLabel8;
QLabel* TextLabel19;
QCheckBox* CheckBox4;
QLabel* TextLabel10;
QLabel* TextLabel14;
QLabel* TextLabel9;
QCheckBox* CheckBox6;
QCheckBox* CheckBox8;
QCheckBox* CheckBox11;
QCheckBox* CheckBox12;
QLabel* TextLabel13;
QLabel* TextLabel16;
QComboBox* ComboBox3;
QPushButton* PushButton14;
QLabel* TextLabel2;
QCheckBox* CheckBox3;
QCheckBox* CheckBox13;
QSpinBox* SpinBox3;
QSpinBox* SpinBox4;
QSpinBox* SpinBox6;
QSpinBox* SpinBox9;
QSpinBox* SpinBox8;
QSpinBox* SpinBox7;
QComboBox* ComboBox2;
QComboBox* ComboBox1;
QSpinBox* SpinBox1;
QSpinBox* SpinBox2;
QSpinBox* SpinBox5;
QComboBox* ComboBox7;
QCheckBox* CheckBox2;
QLineEdit* LineEdit2;
QWidget* tab_2;
QWidget* tab_3;
QWidget* tab_4;
QWidget* tab_5;
QWidget* tab_6;
QMenuBar *menubar;
        QPopupMenu *PopupMenu;
        QPopupMenu *PopupMenu_2;
        QPopupMenu *PopupMenu_3;
        QAction* Action;


    public slots:
        virtual void getFileSlot();
        virtual void addPathSlot();
        virtual void compileFileSlot();
        virtual void removeItem();
        virtual void clearAll();
        virtual void settingsInformation();
        virtual void checkChecked();

    protected:
    };

    #endif // MAINWINDOW_H
```

9.3.3 MainWindow.cpp

```
/* Muhammed Aoun */

#include "MainWindow.h"
#include <qvariant.h>
#include <qcheckbox.h>
#include <qcombobox.h>
#include <qgroupbox.h>
#include <qheader.h>
#include <qlabel.h>
#include <qlineedit.h>
#include <qlistview.h>
#include <qprocess.h>
#include <qprogressbar.h>
#include <qpushbutton.h>
#include <qspinbox.h>
#include <qtabwidget.h>
#include <qtextedit.h>
#include <qwidget.h>
#include <qmime.h>
#include <qdragobject.h>
#include <qlayout.h>
#include <qtooltip.h>
#include <qwhatsthis.h>
#include <qaction.h>
#include <qmenubar.h>
#include <qpopupmenu.h>
#include <qtoolbar.h>
#include <qimage.h>
#include <qpixmap.h>

#include "MainWindow.ui.h"
```

```
static QPixmap uic_load_pixmap_MainWindow( const QString &name )
{
const QMimeSource *m = QMimeSourceFactory::defaultFactory()->data( name );
if ( !m )
return QPixmap();
QPixmap pix;
QImageDrag::decode( m, pix );
return pix;
}
/*
Constructs a MainWindow which is a child of 'parent', with the
name 'name' and widget flags set to 'f'.
*
*/
MainWindow::MainWindow( QWidget* parent,  const char* name, WFlags fl )
: QMainWindow( parent, name, fl )
{
(void)statusBar();
if ( !name )
setName( "MainWindow" );
setEnabled( TRUE );
resize( 697, 552 );
setSizePolicy( QSizePolicy( (QSizePolicy::SizeType)1, (QSizePolicy::SizeType)1, 0, 0, sizePolicy().hasHeightForWidth() ) );
setMinimumSize( QSize( 21, 80 ) );
setMaximumSize( QSize( 697, 552 ) );
setPaletteForegroundColor( QColor( 0, 85, 127 ) );
setPaletteBackgroundColor( QColor( 0, 85, 127 ) );
setCaption( trUtf8( "Frame Manipulation Tools" ) );
setIcon( uic_load_pixmap_MainWindow( "" ) );
setIconText( trUtf8( "" ) );
setCentralWidget( new QWidget( this, "qt_central_widget" ) );

TabWidget5 = new QTabWidget( centralWidget(), "TabWidget5" );
TabWidget5->setGeometry( QRect( 0, 0, 700, 510 ) );
TabWidget5->setPaletteForegroundColor( QColor( 255, 255, 255 ) );
TabWidget5->setPaletteBackgroundColor( QColor( 0, 85, 127 ) );
QFont TabWidget5_font(  TabWidget5->font() );
TabWidget5_font.setFamily( "Times [Urw]" );
TabWidget5_font.setBold( TRUE );
TabWidget5->setFont( TabWidget5_font );

tab = new QWidget( TabWidget5, "tab" );

ProgressBar1 = new QProgressBar( tab, "ProgressBar1" );
```

```
ProgressBar1->setGeometry( QRect( 358, 454, 200, 22 ) );
ProgressBar1->setPaletteBackgroundColor( QColor( 220, 220, 220 ) );

TextLabel1_2 = new QLabel( tab, "TextLabel1_2" );
TextLabel1_2->setGeometry( QRect( 38, 455, 201, 21 ) );
TextLabel1_2->setPaletteForegroundColor( QColor( 255, 255, 255 ) );
TextLabel1_2->setPaletteBackgroundColor( QColor( 0, 85, 127 ) );
QFont TextLabel1_2_font(  TextLabel1_2->font() );
TextLabel1_2_font.setFamily( "Ikarus Vulture" );
TextLabel1_2_font.setItalic( TRUE );
TextLabel1_2->setFont( TextLabel1_2_font );
TextLabel1_2->setFrameShape( QLabel::NoFrame );
TextLabel1_2->setFrameShadow( QLabel::Plain );
TextLabel1_2->setText( trUtf8( "ACIGA" ) );
TextLabel1_2->setAlignment( int( QLabel::AlignCenter ) );

GroupBox9 = new QGroupBox( tab, "GroupBox9" );
GroupBox9->setGeometry( QRect( 10, 340, 681, 110 ) );
GroupBox9->setPaletteForegroundColor( QColor( 255, 255, 255 ) );
GroupBox9->setTitle( trUtf8( "Messages" ) );

TextEdit2 = new QTextEdit( GroupBox9, "TextEdit2" );
TextEdit2->setGeometry( QRect( 10, 20, 661, 77 ) );

GroupBox4 = new QGroupBox( tab, "GroupBox4" );
GroupBox4->setGeometry( QRect( 10, 199, 680, 140 ) );
GroupBox4->setPaletteForegroundColor( QColor( 255, 255, 255 ) );
GroupBox4->setTitle( trUtf8( "List of Single Column Ascii Files to Convert" ) );

PushButton8 = new QPushButton( GroupBox4, "PushButton8" );
PushButton8->setGeometry( QRect( 590, 90, 80, 30 ) );
PushButton8->setText( trUtf8( "Process List" ) );

PushButton11 = new QPushButton( GroupBox4, "PushButton11" );
PushButton11->setGeometry( QRect( 590, 20, 80, 20 ) );
PushButton11->setText( trUtf8( "Remove Item" ) );

PushButton9 = new QPushButton( GroupBox4, "PushButton9" );
PushButton9->setGeometry( QRect( 590, 50, 80, 16 ) );
PushButton9->setText( trUtf8( "Clear All" ) );

ListView1 = new QListView( GroupBox4, "ListView1" );
ListView1->addColumn( trUtf8( "Path                                                  " ) );
ListView1->header()->setClickEnabled( FALSE, ListView1->header()->count() - 1 );
ListView1->addColumn( trUtf8( "Channel" ) );
```

```
ListView1->header()->setClickEnabled( FALSE, ListView1->header()->count() - 1 );
ListView1->addColumn( trUtf8( "Type" ) );
ListView1->header()->setClickEnabled( FALSE, ListView1->header()->count() - 1 );
ListView1->addColumn( trUtf8( "Sampling" ) );
ListView1->header()->setClickEnabled( FALSE, ListView1->header()->count() - 1 );
ListView1->addColumn( trUtf8( "Bits" ) );
ListView1->header()->setClickEnabled( FALSE, ListView1->header()->count() - 1 );
ListView1->addColumn( trUtf8( "Detector" ) );
ListView1->header()->setClickEnabled( FALSE, ListView1->header()->count() - 1 );
ListView1->addColumn( trUtf8( "Compression" ) );
ListView1->header()->setClickEnabled( FALSE, ListView1->header()->count() - 1 );
ListView1->addColumn( trUtf8( "Comp. Level" ) );
ListView1->header()->setClickEnabled( FALSE, ListView1->header()->count() - 1 );
ListView1->addColumn( trUtf8( "Secs/frame" ) );
ListView1->header()->setClickEnabled( FALSE, ListView1->header()->count() - 1 );
ListView1->addColumn( trUtf8( "Limit" ) );
ListView1->header()->setClickEnabled( FALSE, ListView1->header()->count() - 1 );
ListView1->addColumn( trUtf8( "Frames/file" ) );
ListView1->header()->setClickEnabled( FALSE, ListView1->header()->count() - 1 );
ListView1->addColumn( trUtf8( "GPS(secs)" ) );
ListView1->header()->setClickEnabled( FALSE, ListView1->header()->count() - 1 );
ListView1->addColumn( trUtf8( "GPS(nano)" ) );
ListView1->header()->setClickEnabled( FALSE, ListView1->header()->count() - 1 );
ListView1->addColumn( trUtf8( "Skip" ) );
ListView1->header()->setClickEnabled( FALSE, ListView1->header()->count() - 1 );
ListView1->addColumn( trUtf8( "Gain" ) );
ListView1->header()->setClickEnabled( FALSE, ListView1->header()->count() - 1 );
ListView1->setGeometry( QRect( 10, 20, 570, 104 ) );

GroupBox1 = new QGroupBox( tab, "GroupBox1" );
GroupBox1->setGeometry( QRect( 10, 0, 680, 195 ) );
GroupBox1->setPaletteForegroundColor( QColor( 255, 255, 255 ) );
```

```
GroupBox1->setTitle( trUtf8( "Conversion Settings" ) );

TextLabel4 = new QLabel( GroupBox1, "TextLabel4" );
TextLabel4->setGeometry( QRect( 184, -42, 66, 21 ) );
TextLabel4->setText( trUtf8( "TextLabel4" ) );

TextLabel5 = new QLabel( GroupBox1, "TextLabel5" );
TextLabel5->setGeometry( QRect( 11, 86, 65, 30 ) );
TextLabel5->setPaletteForegroundColor( QColor( 255, 255, 255 ) );
TextLabel5->setText( trUtf8( "Secs/frame" ) );

TextLabel11 = new QLabel( GroupBox1, "TextLabel11" );
TextLabel11->setGeometry( QRect( 12, 55, 50, 22 ) );
TextLabel11->setPaletteForegroundColor( QColor( 255, 255, 255 ) );
TextLabel11->setText( trUtf8( "Sampling" ) );

TextLabel15 = new QLabel( GroupBox1, "TextLabel15" );
TextLabel15->setGeometry( QRect( 10, 20, 81, 20 ) );
TextLabel15->setPaletteForegroundColor( QColor( 255, 255, 255 ) );
TextLabel15->setText( trUtf8( "Channel Name" ) );

PushButton16 = new QPushButton( GroupBox1, "PushButton16" );
PushButton16->setGeometry( QRect( 330, 163, 20, 21 ) );
PushButton16->setText( trUtf8( "..." ) );

LineEdit2_2 = new QLineEdit( GroupBox1, "LineEdit2_2" );
LineEdit2_2->setGeometry( QRect( 100, 163, 220, 22 ) );

TextLabel1 = new QLabel( GroupBox1, "TextLabel1" );
TextLabel1->setGeometry( QRect( 10, 163, 84, 21 ) );
TextLabel1->setText( trUtf8( "Select data file" ) );

PushButton12 = new QPushButton( GroupBox1, "PushButton12" );
PushButton12->setGeometry( QRect( 370, 163, 110, 21 ) );
PushButton12->setText( trUtf8( "Add Item to List" ) );

CheckBox1 = new QCheckBox( GroupBox1, "CheckBox1" );
CheckBox1->setGeometry( QRect( 500, 163, 160, 21 ) );
CheckBox1->setText( trUtf8( "Return to default settings" ) );

TextLabel6 = new QLabel( GroupBox1, "TextLabel6" );
TextLabel6->setGeometry( QRect( 526, 90, 66, 22 ) );
TextLabel6->setText( trUtf8( "Frames/file" ) );

CheckBox10 = new QCheckBox( GroupBox1, "CheckBox10" );
CheckBox10->setGeometry( QRect( 598, 92, 21, 21 ) );
```

```
CheckBox10->setText( trUtf8( "CheckBox10" ) );

TextLabel12 = new QLabel( GroupBox1, "TextLabel12" );
TextLabel12->setGeometry( QRect( 354, 86, 73, 30 ) );
TextLabel12->setPaletteForegroundColor( QColor( 255, 255, 255 ) );
TextLabel12->setText( trUtf8( "Compression \n"
"Level" ) );

CheckBox9 = new QCheckBox( GroupBox1, "CheckBox9" );
CheckBox9->setGeometry( QRect( 429, 92, 21, 21 ) );
CheckBox9->setText( trUtf8( "CheckBox9" ) );

TextLabel7 = new QLabel( GroupBox1, "TextLabel7" );
TextLabel7->setGeometry( QRect( 172, 90, 73, 22 ) );
TextLabel7->setText( trUtf8( "Compression" ) );

CheckBox7 = new QCheckBox( GroupBox1, "CheckBox7" );
CheckBox7->setGeometry( QRect( 253, 92, 21, 21 ) );
CheckBox7->setText( trUtf8( "CheckBox7" ) );

CheckBox5 = new QCheckBox( GroupBox1, "CheckBox5" );
CheckBox5->setGeometry( QRect( 465, 52, 21, 21 ) );
CheckBox5->setText( trUtf8( "CheckBox5" ) );

TextLabel8 = new QLabel( GroupBox1, "TextLabel8" );
TextLabel8->setGeometry( QRect( 426, 52, 30, 22 ) );
TextLabel8->setPaletteForegroundColor( QColor( 255, 255, 255 ) );
TextLabel8->setText( trUtf8( "Limit" ) );

TextLabel19 = new QLabel( GroupBox1, "TextLabel19" );
TextLabel19->setGeometry( QRect( 289, 51, 30, 22 ) );
TextLabel19->setText( trUtf8( "Gain" ) );

CheckBox4 = new QCheckBox( GroupBox1, "CheckBox4" );
CheckBox4->setGeometry( QRect( 324, 52, 21, 21 ) );
CheckBox4->setText( trUtf8( "CheckBox4" ) );

TextLabel10 = new QLabel( GroupBox1, "TextLabel10" );
TextLabel10->setGeometry( QRect( 165, 54, 25, 22 ) );
TextLabel10->setPaletteForegroundColor( QColor( 255, 255, 255 ) );
TextLabel10->setText( trUtf8( "Bits" ) );

TextLabel14 = new QLabel( GroupBox1, "TextLabel14" );
TextLabel14->setGeometry( QRect( 10, 126, 63, 22 ) );
TextLabel14->setPaletteForegroundColor( QColor( 255, 255, 255 ) );
TextLabel14->setText( trUtf8( "GPS (secs)" ) );
```

```
TextLabel9 = new QLabel( GroupBox1, "TextLabel9" );
TextLabel9->setGeometry( QRect( 569, 50, 26, 22 ) );
TextLabel9->setPaletteForegroundColor( QColor( 255, 255, 255 ) );
TextLabel9->setText( trUtf8( "Skip" ) );

CheckBox6 = new QCheckBox( GroupBox1, "CheckBox6" );
CheckBox6->setGeometry( QRect( 601, 52, 21, 21 ) );
CheckBox6->setText( trUtf8( "CheckBox6" ) );

CheckBox8 = new QCheckBox( GroupBox1, "CheckBox8" );
CheckBox8->setGeometry( QRect( 81, 92, 21, 21 ) );
CheckBox8->setText( trUtf8( "CheckBox8" ) );

CheckBox11 = new QCheckBox( GroupBox1, "CheckBox11" );
CheckBox11->setGeometry( QRect( 81, 128, 21, 21 ) );
CheckBox11->setText( trUtf8( "CheckBox11" ) );

CheckBox12 = new QCheckBox( GroupBox1, "CheckBox12" );
CheckBox12->setGeometry( QRect( 314, 128, 21, 21 ) );
CheckBox12->setText( trUtf8( "CheckBox12" ) );

TextLabel13 = new QLabel( GroupBox1, "TextLabel13" );
TextLabel13->setGeometry( QRect( 247, 126, 64, 22 ) );
TextLabel13->setPaletteForegroundColor( QColor( 255, 255, 255 ) );
TextLabel13->setText( trUtf8( "GPS (nano)" ) );

TextLabel16 = new QLabel( GroupBox1, "TextLabel16" );
TextLabel16->setGeometry( QRect( 274, 20, 31, 22 ) );
TextLabel16->setPaletteForegroundColor( QColor( 255, 255, 255 ) );
TextLabel16->setText( trUtf8( "Type" ) );

ComboBox3 = new QComboBox( FALSE, GroupBox1, "ComboBox3" );
ComboBox3->insertItem( trUtf8( "adc" ) );
ComboBox3->insertItem( trUtf8( "proc" ) );
ComboBox3->insertItem( trUtf8( "sim" ) );
ComboBox3->setGeometry( QRect( 314, 20, 80, 22 ) );

PushButton14 = new QPushButton( GroupBox1, "PushButton14" );
PushButton14->setGeometry( QRect( 455, 20, 190, 21 ) );
PushButton14->setText( trUtf8( "Information About These Settings" ) );

TextLabel2 = new QLabel( GroupBox1, "TextLabel2" );
TextLabel2->setGeometry( QRect( 493, 128, 52, 21 ) );
TextLabel2->setPaletteForegroundColor( QColor( 255, 255, 255 ) );
TextLabel2->setText( trUtf8( "Detector" ) );
```

```
CheckBox3 = new QCheckBox( GroupBox1, "CheckBox3" );
CheckBox3->setGeometry( QRect( 195, 54, 21, 21 ) );
CheckBox3->setText( trUtf8( "CheckBox3" ) );

CheckBox13 = new QCheckBox( GroupBox1, "CheckBox13" );
CheckBox13->setGeometry( QRect( 554, 128, 21, 21 ) );
CheckBox13->setText( trUtf8( "CheckBox13" ) );
CheckBox13->setChecked( TRUE );

SpinBox3 = new QSpinBox( GroupBox1, "SpinBox3" );
SpinBox3->setEnabled( FALSE );
SpinBox3->setGeometry( QRect( 103, 54, 40, 22 ) );
QPalette pal;

pal.setDisabled( cg );
ComboBox7->setPalette( pal );

CheckBox2 = new QCheckBox( GroupBox1, "CheckBox2" );
CheckBox2->setGeometry( QRect( 81, 56, 21, 21 ) );
CheckBox2->setText( trUtf8( "CheckBox2" ) );

LineEdit2 = new QLineEdit( GroupBox1, "LineEdit2" );
LineEdit2->setGeometry( QRect( 102, 20, 130, 22 ) );
LineEdit2->setFocusPolicy( QLineEdit::StrongFocus );
LineEdit2->setMaxLength( 20 );
TabWidget5->insertTab( tab, trUtf8( "To Frames - Single Ascii" ) );

tab_2 = new QWidget( TabWidget5, "tab_2" );
TabWidget5->insertTab( tab_2, trUtf8( "To Frames - Ascii table" ) );

tab_3 = new QWidget( TabWidget5, "tab_3" );
TabWidget5->insertTab( tab_3, trUtf8( "Split Frames" ) );

tab_4 = new QWidget( TabWidget5, "tab_4" );
TabWidget5->insertTab( tab_4, trUtf8( "Merge/Copy" ) );

tab_5 = new QWidget( TabWidget5, "tab_5" );
TabWidget5->insertTab( tab_5, trUtf8( "Output to Screen" ) );

tab_6 = new QWidget( TabWidget5, "tab_6" );
TabWidget5->insertTab( tab_6, trUtf8( "Integrity Check" ) );

// actions
Action = new QAction( this, "Action" );
Action->setText( trUtf8( "Action" ) );
```

```
// toolbars

// menubar
menubar = new QMenuBar( this, "menubar" );

PopupMenu = new QPopupMenu( this );
menubar->insertItem( trUtf8( "&File" ), PopupMenu );

PopupMenu_2 = new QPopupMenu( this );
menubar->insertItem( trUtf8( "&System Settings" ), PopupMenu_2 );

PopupMenu_3 = new QPopupMenu( this );
menubar->insertItem( trUtf8( "&Help" ), PopupMenu_3 );

// signals and slots connections
connect( PushButton16, SIGNAL( clicked() ), this, SLOT( getFileSlot() ) );
connect( PushButton12, SIGNAL( clicked() ), this, SLOT( addPathSlot() ) );
connect( PushButton8, SIGNAL( clicked() ), this, SLOT( compileFileSlot() ) );
connect( PushButton14, SIGNAL( clicked() ), this, SLOT( settingsInformation() )
);
connect( PushButton11, SIGNAL( clicked() ), this, SLOT( removeItem() ) );
connect( PushButton9, SIGNAL( clicked() ), this, SLOT( clearAll() ) );
connect( CheckBox2, SIGNAL( clicked() ), this, SLOT( checkChecked() ) );
connect( CheckBox3, SIGNAL( clicked() ), this, SLOT( checkChecked() ) );
connect( CheckBox4, SIGNAL( clicked() ), this, SLOT( checkChecked() ) );
connect( CheckBox5, SIGNAL( clicked() ), this, SLOT( checkChecked() ) );
connect( CheckBox6, SIGNAL( clicked() ), this, SLOT( checkChecked() ) );
connect( CheckBox8, SIGNAL( clicked() ), this, SLOT( checkChecked() ) );
connect( CheckBox7, SIGNAL( clicked() ), this, SLOT( checkChecked() ) );
connect( CheckBox9, SIGNAL( clicked() ), this, SLOT( checkChecked() ) );
connect( CheckBox10, SIGNAL( clicked() ), this, SLOT( checkChecked() ) );
connect( CheckBox11, SIGNAL( clicked() ), this, SLOT( checkChecked() ) );
connect( CheckBox12, SIGNAL( clicked() ), this, SLOT( checkChecked() ) );
connect( CheckBox13, SIGNAL( clicked() ), this, SLOT( checkChecked() ) );
}

/*
Destroys the object and frees any allocated resources
*/
MainWindow::~MainWindow()
{
```

```
// no need to delete child widgets, Qt does it all for us
}
```

9.3.4 MainWindow.ui.h

```
/****************************************************************
** ui.h extension file, included from the uic-generated form implementation.
**
** If you wish to add, delete or rename slots use Qt Designer which will
** update this file, preserving your code. Create an init() slot in place of
** a constructor, and a destroy() slot in place of a destructor.
*****************************************************************/
#include <qfiledialog.h>
#include <qfiledlg.h>
#include <qlistview.h>
#include <qmessagebox.h>
#include <qprocess.h>
#include <qfileinfo.h>
#include <qvalidator.h>
#include <ctype.h>
#include <qlineedit.h>
#include <qregexp.h>
#include <qapplication.h>

QString type1;
QString channel1;
QString compression1;
QString compressionLevel1;
QString detector1;
QString path1;
QString sampling1;
QString secsFrame1;
QString framesFile1;
QString gpsSec1;
QString gpsNano1;
QString bits1;
QString limit1;
QString skip1;
QString gain1;
int choice;
QRegExp rx("[\\w]");//abcdefghijklmnopqrstuvwxyz1234567890_
QRegExpValidator validator(rx,0);

void MainWindow::getFileSlot()
```

```
{
QString filename = QFileDialog::getOpenFileName("/","*.*");
LineEdit2_2 -> setText (filename);
}

void MainWindow::addPathSlot()
{


type1 = ComboBox3 -> currentText();
channel1 = LineEdit2 -> text();
path1 = LineEdit2_2->text();

if ( CheckBox7->isChecked() == TRUE)
{
compression1 = ComboBox2 -> currentText();
}
else
{
compression1 = "-";
}

if ( CheckBox9->isChecked() == TRUE)
{
compressionLevel1 = ComboBox1 -> currentText();
}
else
{
compressionLevel1 = "-";
}

if ( CheckBox13->isChecked() == TRUE)
{
detector1 = ComboBox7 -> currentText();
}
else
{
detector1 = "-";
}

if ( CheckBox2->isChecked() == TRUE)
{
sampling1 = SpinBox3 -> cleanText();
}
else
{
```

```
sampling1 = "-";
}

if ( CheckBox8->isChecked() == TRUE)
{
secsFrame1 = SpinBox7-> cleanText();
}
else
{
secsFrame1 = "-";
}

if ( CheckBox10->isChecked() == TRUE)
{
framesFile1 = SpinBox1 -> cleanText();
}
else
{
framesFile1 = "-";
}

if ( CheckBox11->isChecked() == TRUE)
{
gpsSec1 = SpinBox2 -> cleanText();
}
else
{
gpsSec1 = "-";
}

if ( CheckBox12->isChecked() == TRUE)
{
gpsNano1 = SpinBox5 -> cleanText();
}
else
{
gpsNano1 = "-";
}

if ( CheckBox3->isChecked() == TRUE)
{
bits1 = SpinBox4 -> cleanText();
}
else
{
bits1 = "-";
```

```
}

if ( CheckBox5->isChecked() == TRUE)
{
limit1 = SpinBox9 -> cleanText();
}
else
{
limit1 = "-";
}

if ( CheckBox6->isChecked() == TRUE)
{
skip1 = SpinBox8 -> cleanText();
}
else
{
skip1 = "-";
}

if ( CheckBox4->isChecked() == TRUE)
{
gain1 = SpinBox6 -> cleanText();
}
else
{
gain1 = "-";
}
if ( channel1.isEmpty() == FALSE )
{
QFileInfo fileInQuestion(path1);
if (fileInQuestion.isFile())
{
QListViewItem *item = new QListViewItem( ListView1);

item->setText(0,path1);//path
item->setText(1,channel1);//channel
item->setText(2,type1);//type
item->setText(3,sampling1);//sampling
item->setText(4,bits1);//bits
item->setText(5,detector1);//detector
item->setText(6,compression1);//compression
item->setText(7,compressionLevel1);//compression rate
item->setText(8,secsFrame1);//secs/frame
item->setText(9,limit1);//limit
item->setText(10,framesFile1);//frame file
item->setText(11,gpsSec1);//gps(secs)
item->setText(12,gpsNano1);//gps(nano)
item->setText(13,skip1);//skips
item->setText(14,gain1);//gain

int count;
count = (ListView1-> childCount());
SpinBox1 -> setValue(count);
if ( CheckBox1->isChecked() == TRUE)
{
ComboBox3 -> setCurrentText("adc");//type
LineEdit2 -> setText("");//channel
LineEdit2_2->setText("/");//path
ComboBox2 -> setCurrentText("3");//compression
ComboBox1 -> setCurrentText("1");//compression level
ComboBox7 -> setCurrentText("ACIGA");//detector
SpinBox3 -> setValue(0);//sampling
SpinBox7-> setValue(1);//secs per frame
SpinBox1 -> setValue(1);//frames per file
SpinBox2 -> setValue(0);//gps secs
SpinBox5 -> setValue(0);//gps nano
SpinBox4 -> setValue(-32);//bits
SpinBox9 -> setValue(0);//limit
SpinBox8 -> setValue(0);//skip
SpinBox6 -> setValue(0);//gain
}
}

else
{
QMessageBox::warning (0, "File Error", "Sorry, the file you selected is not valid.", "OK", "", QString::null, 0, 1);
}
}

else
{
QMessageBox::warning (0, "Channel Error", "Please enter a valid channel name.", "OK", "", QString::null, 0, 1);
}

}

void MainWindow::compileFileSlot()
{
int x = 0;
```

```
while ( x < ListView1->childCount())
{
QListViewItem *itemf = new QListViewItem( ListView1);
QString bllah;
bllah = itemf->text(1);
TextEdit2->insert(bllah);
/*QProcess *proc;
proc = new QProcess( this );

proc->addArgument( "/ligoapps/ligotools/bin/ascii2frame" );
proc->addArgument( "-channel" );
proc->addArgument( "channel1" );
proc->addArgument( "-i" );
proc->addArgument( "/ligoapps/ligotools/bin/ascii_sample1.data" );
proc->addArgument( "-type" );
proc->addArgument( "adc" );
proc->start();*/
//proc->kill();
//QCString five = 5; //This line converts an int (5) to a string.
//QMessageBox::warning (0, "Channel Error", "Please enter a valid
channel name.", "OK", "", QString::null, 0, 1);
x++;
}

}

void MainWindow::removeItem()
{
QListViewItem* i = ListView1->currentItem();
if (ListView1->isSelected(i))
ListView1->removeItem(i);
}

void MainWindow::clearAll()
{
choice = QMessageBox::warning (0, "Clear All", "Are you sure you want
to clear the whole list?", "Yes", "No", QString::null, 0, 1);

if (choice == 0)
ListView1->clear();
}

void MainWindow::settingsInformation()
{
QString channel10;
```

```
channel10 = "abcdefg";//LineEdit2->text()
LineEdit2->setText(channel10);
/*int ret = rx.search(channel10);
if (ret != -1)
QMessageBox::warning (0, "Clear All", "FALSE", "OK", "", QString::null, 0, 1);
else
QMessageBox::warning (0, "Clear All", "TRUE", "OK", "", QString::null, 0, 1);*/
}


void MainWindow::checkChecked()
{

if ( CheckBox2->isChecked() == TRUE)
{
SpinBox3->setEnabled(TRUE);
}
else
{
SpinBox3->setEnabled(FALSE);
}

if ( CheckBox3->isChecked() == TRUE)
{
SpinBox4->setEnabled(TRUE);
}
else
{
SpinBox4->setEnabled(FALSE);
}

if ( CheckBox4->isChecked() == TRUE)
{
SpinBox6->setEnabled(TRUE);
}
else
{
SpinBox6->setEnabled(FALSE);
}

if ( CheckBox5->isChecked() == TRUE)
{
SpinBox9->setEnabled(TRUE);
}
```

```
else
{
SpinBox9->setEnabled(FALSE);
}

if ( CheckBox6->isChecked() == TRUE)
{
SpinBox8->setEnabled(TRUE);
}
else
{
SpinBox8->setEnabled(FALSE);
}

if ( CheckBox7->isChecked() == TRUE)
{
ComboBox2->setEnabled(TRUE);
}
else
{
ComboBox2->setEnabled(FALSE);
}

if ( CheckBox8->isChecked() == TRUE)
{
SpinBox7->setEnabled(TRUE);
}
else
{
SpinBox7->setEnabled(FALSE);
}

if ( CheckBox9->isChecked() == TRUE)
{
ComboBox1->setEnabled(TRUE);
}
else
{
ComboBox1->setEnabled(FALSE);
}

if ( CheckBox10->isChecked() == TRUE)
{
SpinBox1->setEnabled(TRUE);
}
else
```

```
{
SpinBox1->setEnabled(FALSE);
}

if ( CheckBox11->isChecked() == TRUE)
{
SpinBox2->setEnabled(TRUE);
}
else
{
SpinBox2->setEnabled(FALSE);
}

if ( CheckBox12->isChecked() == TRUE)
{
SpinBox5->setEnabled(TRUE);
}
else
{
SpinBox5->setEnabled(FALSE);
}

if ( CheckBox13->isChecked() == TRUE)
{
ComboBox7->setEnabled(TRUE);
}
else
{
ComboBox7->setEnabled(FALSE);
}

}
```

9.4 Components Relationship Overview

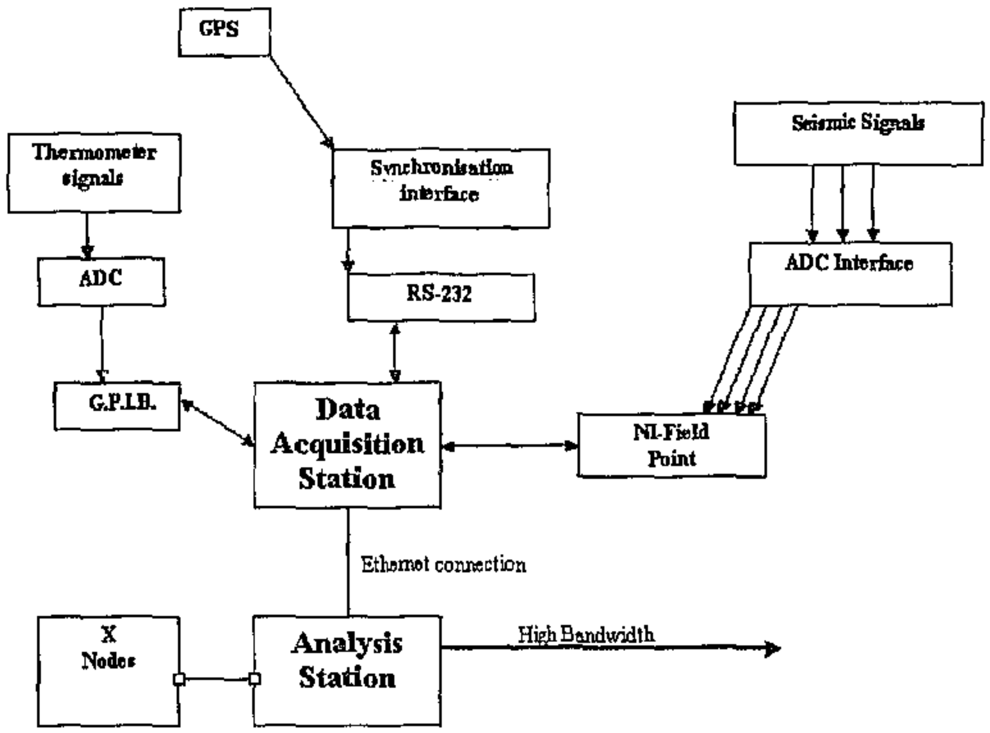


Figure 9.4.1 Data acquisition hardware system.

Data Acquisition Station: The computer where all data gathering instruments and their interfaces is connected.

Analysis Station: Data from the Data Acquisition Station is sent to the Analysis Station for storage, analysis and conversion into the frame format. This separate system is needed so no system resources are taken that may slow down Data Acquisition. The Analysis station will be online and access give for other detector sites to request data any time needed.

Seismic Signals: These include wind signals, electro-magnetic signals etc. Basically any seismic even that may affect the gravity wave signal.

GPS: The GPS time is needed to be recorded simultaneously with the data gathered from the Seismic signals. This is for the purposes of allowing other observatories to compare what data they had gathered at the exact same time.

9.5 Collaborators

AIGO (Australian International Gravitational Observatory)

- http://www.gravity.uwa.edu.au
- Locations is in Gingin, Western Australia
- Institutes involved in the Australian project include:
    - University of Western Australia
    - The Australian National University
    - University of Adelaide
    - Monash University

LIGO (Laser Interferometer Gravitational Wave Observatory)

- http://www.ligo.caltech.edu/
- Location is in Livingston, U.S.A.
- Institutes part of the U.S. project include:
    - California Institute of Technology
    - Massachusetts Institute of Technology
    - NASA (Project LISA: Laser Interferometer Space Antenna)
        - http://lisa.jpl.nasa.gov/

VIRGO (European Gravity Wave observatory)

- http://www.virgo.infn.it/
- Virgo is located at Cascina, near Pisa on the Arno plain, Italy.
- Collaboration between both France and Italy.

TAMA (Japanese Gravity Wave observatory)

- http://tamago.mtk.nao.ac.jp/
- TAMA is located at Saitama, Japan

9.6 Definitions and Abbreviations

ACIGA = Australian Consortium for Interferometric Gravitational Astronomy.

AIGO = Australian International Gravitational Observatory.

GPS time-stamping = A GPS satellite which omits a global time in nanoseconds. A satellite omits pulses giving the same number of nanoseconds any where on earth.

Gravity waves = A hypothetical wave that is held to propagate the force of gravity and to travel at the speed of light, also called gravity wave.

Gingin = Large district located 1 hour north of Perth.

Interferometer = "Any of several optical, acoustic, or radio frequency instruments that use interference phenomena between a reference wave and an experimental wave or between two parts of an experimental wave to determine wavelengths and wave velocities, measure very small distances and thicknesses, and calculate indices of refraction."[16]

LIGO = Laser Interferometer Gravitational Observatory

Linux = An Operating System which simulates the UNIX system, and can be run on Intel or Motorola CPU machines.

Seismic data = Data of which represents the measurement of environmental phenomena extracted from seismic instruments.

TAMA = Gravity wave detector site in Japan.

VIRGO = Variability of solar Infrared-Radiance and Gravity Oscillations

[16] http://www.dictionary.com

9.7 References

California Institute of Technology. Laser Interferometer Gravitational Wave Observatory [on-line] Available WWW: http://www.ligo.caltech.edu/ [January, 2003]

Dalheimer, M.K.(2002) Programming with QT, 2nd Edition O'Rielly: New York

Hammel, M.J. Linux Magazine: The History of XFree86 - Over a Decade of Development [on-line] Available WWW: http://www.linux-mag.com/2001-12/xfree86_01.html [Dec, 2001]

Hubbard, J. (2000) Programming with C++ 2nd Edition McGraw-Hill: New York

Lexico LLC, Dictionary.com [on-line]. Available WWW: http://www.dictionary.com [June 2002]

Queru, E. Book Review: Programming with Qt, 2nd Edition [on-line] Available WWW: http://www.osnews.com/story.php?news_id=745 [Mar, 2002]

TAMA. The 300m Laser Interferometer Gravitational Wave Antenna [on-line] Available WWW: http://tamago.mtk.nao.ac.jp/ [December, 2002]

Trolltech Technologies. QT Reference Documentation [on-line] Available WWW: http://doc.trolltech.com/3.1/ [January, 2003]

Trolltech Technologies. Trolltech Wins "Best Embedded Solution" for Second Straight Year [on-line] Available WWW: http://www.trolltech.com/company/announce.html?Action=Show&AID=104 [August, 2002]

Trolltech Technologies. Qt 2.0 Wins LinuxWorld Award [on-line] Available WWW: http://www.trolltech.com/company/announce.html?Action=Show&AID=60 [August, 1999]

University of Western Australia. Australian International Gravitational Research Centre [on-line] Available WWW: http://www.gravity.uwa.edu.au [December, 2002]

Vicere, A. Frame builder emulation (Fbe) library v1r2 [on-line] Available WWW: http://www.ldas-sw.ligo.caltech.edu/ligotools/Fbe/html/ [January, 2003]

VIRGO. VIRGO Project Central Web Site [on-line] Available WWW: http://www.virgo.infn.it/ [November, 2002]

VIRGO-SPE-LAP-5400-102, Specification of a Common Data Frame Format for Interferometric Gravitational Wave Detectors (IGWD [Available WWW:

http://wwwlapp. in2p3. fr/virgo/FrameL/T970130-D-E.pdf [Dec, 2000]

VIR-MAN-LAP-5400-103, Frame Library (Fr) Users Manual [on-line]. Available WWW: http://wwwlapp.in2p3.fr/virgo/FrameL/FrDoc.html [May, 2002]

9.8 Change Log

Current version - version 2.5 (February 2003)

Major revisions for version 2:

- Minor spelling errors fixed throughout.
- Reformatted Significance and Purpose sections. Combined them into one (section 2.2)
- Included a "Previous studies in this field" section which was missing but critical. (section 2.5)
- Included new section QT-Interest group. (section 2.4)
- Change log (section 2.6)
- Removed foreign findings from conclusion section.
- Updated referencing; used ECU standard.
- Editing and minor grammar corrections.
- Total revamp of section 7.2. Added information concerning user evaluation of GUI, and how such evaluation occurred.

Major revisions for version 1 (October 2002):

- Spelling errors throughout document corrected, Australian vs. USA spelling issues.
- Purpose of study statement re-evaluated, added the problem of large amounts of data needing to undergo an automated process. (section 2.3)
- Included a calculation example of vast data amount. (section 2.3)
- Several more research questions have been added, mainly in respect to design and efficiency metrics issues (section 2.4)
- Complete overhaul of literature review, greater in-depth analysis. (section 3.0)

- More detailed information concerning the instruments and procedures to carry out the research with reference to an action research approach explained further. (section 4.3)
- Change log (s ction 5.6)