A Java Swing Applet for Physiological Signal and Annotation Display

by

Emily A. Liu

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUL 3 1 2002
LIBRARIES

Submitted to the Department of Electrical Engineering and Computer Science BARKER

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Computer Science and Engineering

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

August 13, 2001

The author hereby grants to M.I.T. permission to reproduce and distribute publicly paper and electronic copies of this thesis and grant others the right to do so.

Author_____
Department of Electrical Engineering and Computer Science
August 13, 2001

Certified by_____
Roger G. Mark, M.D., Ph.D.
Professor of EECS

Accepted by_____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

A Java Swing Applet for Physiologic Signal and Annotation Display

by
Emily A. Liu

Submitted to the
Department of Electrical Engineering and Computer Science

August 13, 2001

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

This project introduces a Java based Swing applet viewing system that gives users the ability to visually browse through recorded physiologic data and annotations. The focus is to fulfill a particular need within the Research Resource for Physiologic Signals Toolkit signal viewing resources in providing a simple signal and annotation viewer that utilizes the power of an immediate client-side web software application. Users of this Java software system are able to view archived physiologic data as waveforms and annotation data within a web browser on major platforms. This signal and annotation viewer has capabilities including basic waveform and annotation display, full record browsing, time search, forward and backward annotation search, time rescaling, and amplitude rescaling.

Thesis Supervisor: Roger G. Mark, M.D., Ph.D
Title: Distinguished Professor in Health Sciences and Technology, HST
Professor of Electrical and Bioengineering, Department of Electrical Engineering and Computer Science, MIT

TABLE OF CONTENTS

1 INTRODUCTION 6

2 BACKGROUND 7

2.1 RESEARCH RESOURCE FOR COMPLEX PHYSIOLOGIC SIGNALS 7
2.1.1 PhysioBank 8
2.1.2 PhysioToolkit 9
2.1.3 PhysioNet 10
2.2 WAVE – A SIGNAL AND ANNOTATION EDITOR 11

3 REQUIREMENTS AND SPECIFICATIONS 12

3.1 OVERVIEW 12
3.2 APPLET SPECIFICATIONS 13
3.2.1 Graphical User Interface (GUI) requirements 13
3.2.2 Signal and Annotation Storge 14
3.3 USER'S MANUAL 15
3.3.1 Loading the Applet 15
3.3.2 Navigation Control 16
3.3.3 Toggles 18
3.3.4 Search By Annotation 19
3.3.5 Magnification 21
3.3.6 Exiting the Applet 22
3.4 PERFORMANCE 22

4 DESIGN 23

4.1 CRITERIA 24
4.1.1 Portable 24
4.1.2 Intuitive 24
4.1.3 Time efficient 25
4.1.4 Modular 25
4.1.5 Extensible 26
4.1.6 Scalable 26
4.2 IMPLEMENTATION 26
4.2.1 Java Platform 28
4.2.2 Java Foundation Classes - Swing 29
4.3 MODULE STRUCTURE 30
4.3.1 GUI Modules 31
4.3.2 Data Structures 32
4.3.3 Data Reading Objects 33
4.3.4 GraphProcess 34

5 TESTING 35

6 REFLECTION 36

6.1 EVALUATION 36
6.2 KEY LESSONS 38

7 CONCLUSION 39

BIBLIOGRAPHY 40

APPENDIX A – FORMATS 41

APPENDIX B – MODULE DEPENDENCY DIAGRAM 42

APPENDIX C – MODULE SPECIFICATIONS 43

APPENDIX D – JAVA CODE

TABLE OF FIGURES

FIGURE 1 *PHYSIOBANK* WEBPAGE 8

FIGURE 2 TABLE OF DATABASES WITHIN *PHYSIOBANK* 9

FIGURE 3 *PHYSIOTOOLKIT* WEBPAGE 9

FIGURE 4 *PHYSIONET* WEBPAGE 10

FIGURE 5 WAVE SCREENSHOT WITH RECORD 100 FROM MITDB 12

FIGURE 6 LIST OF URLS SPECIFYING RECORD/ANNOTATION COMBINATIONS 16

FIGURE 7 APPLET WINDOW WITH NAVIGATION CONTROLS CIRCLED 17

FIGURE 8 "GO TO" DIALOG BOX 17

FIGURE 9 APPLET WINDOW WITH TOGGLE OPTIONS CIRCLED 18

FIGURE 10 APPLET WINDOW WITH ANNOTATION SEARCH BUTTONS CIRCLED 19

FIGURE 11 ANNOTATION SEARCH DIALOG BOX 20

FIGURE 12 ANNOTATION FOUND 20

FIGURE 13 APPLET WINDOW WITH TIME AND AMPLITUDE SCALE MENUS CIRCLED 21

FIGURE 14 CHART-O-MATIC 27

FIGURE 15 FORTE FOR JAVA IDE 28

ACKNOWLEDGEMENTS

Professor Roger Mark, M.D., Ph.D.
George Moody
Isaac Henry
Christopher Cheng
Richard Perng

1 Introduction

One of the most influential technologies throughout the world is the Internet and the World Wide Web. The ability to harness electronic information has not only changed the way people communicate, but has also broadened the way in which they store and gather information. The advent of this technological era has ushered in changes within the biomedical community as well. In an effort to take advantage of this information communication tool, the Research Resource for Complex Physiologic Signals[1] has been established. Its purpose is to facilitate a better exchange and presentation of information within the biomedical community.

This particular thesis focuses on fulfilling a particular need within the Research Resource Toolkit signal viewing resources. Although more complex signal and annotation viewers and editors already exist within this resource, no current implementations utilize the power of an immediate client-side web software application and/or the power of object-oriented programming. The purpose of this thesis project is to create a new Java software system that allows users to view archived physiologic data as waveforms and annotation data within a web browser using the power of client-side[2] computation. This signal and annotation viewer has capabilities including basic waveform and annotation display, full record browsing, time search, forward and backward annotation search, time rescaling, and amplitude rescaling.

In this paper, section two will examine both the background of the Research Resource for Complex Physiologic Signals and the specific impetus for developing a new

---

[1] Funded by the National Center for Research Resources of the National Institutes of Health, grant.

[2] Client-side: A client is the requesting program or user in a client/server relationship. For example, a Web browser has a client relationship with the computer that is getting and returning a requested HTML file. Client-side refers to any activity on this client computer.

software viewer. Section three, the requirements section, will present a detailed description of the problem and the solution. This section will also contain a user's manual and performance analysis. Section four will be dedicated to topics relating to the design of the software system including a high-level picture of implementation strategy and an explanation of specific design decisions. The fifth section will present testing strategies used to verify the correctness of this software program and the results. Section six will contain an evaluation of both the software system and its development process. The final seventh section will discuss follow-up research. The appendix will include low-level details about the system including anything pertaining to actual code.

2 Background

2.1 Research Resource for Complex Physiologic Signals

In the biomedical field, each research group tends to be autonomous in gathering data and devising tools for signal analysis. Although researchers gather periodically to discuss issues and publish papers to disseminate information, several tasks such as data gathering may be repeated among groups researching similar topics. Researchers typically invest considerable time and money to collect data and to develop signal processing software when similar databases and software modules may already exist as a result of the efforts of other groups. For these reasons, it would be of considerable benefit to the research community to share both data and software. It is only by using common or standardized databases that researchers can compare and evaluate their results. The Research Resource for Complex Physiologic Signals[3] is an effort to provide

[3] Goldberger, et al.

an extensive set of common resources that includes databases and signal processing software. The goal of the Resource is to stimulate sharing of resources and the advancement of solutions for hard-to-diagnose medical problems. The availability of this resource also helps create standardization of data and allows validation by peer review. In addition, the Resource facilitates a concentration of research efforts focused on specific signal processing challenges[4] (see section 2.1.3 *PhysioNet*.)

The Research Resource for Complex Physiologic Signals is comprised of three parts: *PhysioBank* – a collection of physiologic databases, *PhysioToolkit* – a set of software modules and signal processing tools, and *PhysioNet* – an online forum to collect and disseminate information.

2.1.1 PhysioBank

PhysioBank is an archive of the Research Resource that holds a large and growing collection of physiologic recordings and related data, including well-characterized signals, raw data supporting published journal articles, and other contributed collections of data. These contributed collections include works in progress, for use by the biomedical research community.[5] At present, the data files already posted occupy approximately 30 gigabytes of information. This archive provides databases of multi-parameter cardiopulmonary, neural, and other biomedical signals as a resource to

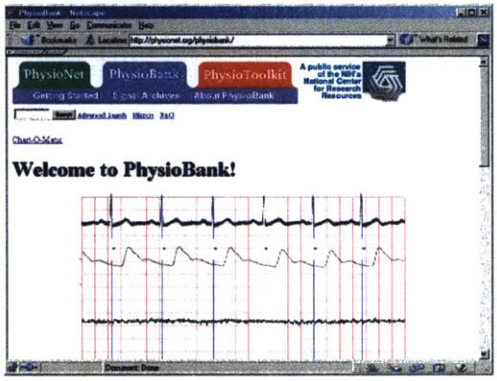


Figure 1 *PhysioBank* webpage.

[4] "Detecting and quantifying sleep apnea…"
[5] Goldberger, et al.

aid and encourage further investigation of pathological conditions that have major public health implications. These conditions include epilepsy, congestive heart failure, sleep apnea, sudden cardiac death, myocardial infarction, gait disorders, and aging. Figure 1 is a snapshot of the *PhysioBank* webpage. Figure 2 is a table of all databases within *PhysioBank.*

Apnea-ECG Database (apnea-ecg)
BIDMC Congestive Heart Failure Database (chfdb)
CU Ventricular Tachyarrhythmia Database (cudb)
European ST-T Database (edb)
Fantasia Database (fantasia)
Gait in Aging and Disease Database (gaitdb)
Gait in Neurodegenerative Disease Database (gaitndd)
Gait Maturation Database (gait-maturation-db/data)
Heart Rate Oscillations during Meditation (meditation/data)
Heart Rate Oscillations in Partial Epilepsy (szdb)
Long Term ST Database (ltstdb)
MIMIC Database (mimicdb)
MIT-BIH Arrhythmia Database (mitdb)
MIT-BIH Atrial Fibrillation Database (afdb)
MIT-BIH ECG Compression Test Database (cdb)
MIT-BIH Long-Term ECG Database (ltdb)
MIT-BIH Malignant Ventricular Ectopy Database (vfdb)
MIT-BIH Noise Stress Test Database (nstdb)
MIT-BIH Normal Sinus Rhythm Database (nsrdb)
... [Recordings excluded from the NSR DB] (excluded)
MIT-BIH Polysomnographic Database (slpdb)
MIT-BIH ST Change Database (stdb)
MIT-BIH Supraventricular Arrhythmia Database (svdb)
PAF Prediction Challenge Database (afpdb)
QT Database (qtdb)
Samples of other databases (odb)

Figure 2 Table of databases within *PhysioBank*

2.1.2 PhysioToolkit

PhysioToolkit contains a library of software for manipulating physiologic data. The software includes tools for signal processing, signal analysis, and detection of physiologically significant events using both classical techniques and novel methods based on statistical physics and nonlinear dynamics.[6] All research projects that contribute to the *PhysioToolkit* have the common

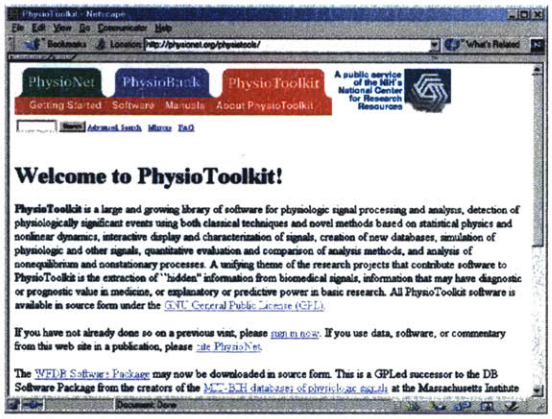


Figure 3 *PhysioToolKit* webpage.

---

[6] ibid.

goal of extracting "hidden" information from biomedical signals. This information may provide either diagnostic and prognostic value in medicine or explanatory and predictive power in basic research. *PhysioToolkit* also contains software that contributes to aiding further research on physiologic data such as signal and annotation viewers and editors. All of the software is available under the GNU General Public License (GPL). Figure 3 is a snapshot of the PhysioToolKit webpage.

2.1.3 PhysioNet

The third section of the Research Resource provides free electronic access to the rest of the Resource (*PhysioBank* and *PhysioToolkit*) and other non-fully supported contributed materials via the World Wide Web. *PhysioNet* is an online forum that assists in disseminating and exchanging biomedical signals and open-source signal analysis software. This forum allows for efficient retrieval and submission of data and software and provides the facilities to help stimulate cooperative analysis of data and evaluation of new proposed algorithms. *PhysioNet* also co-hosts the annual *Computers in Cardiology*[7] contest which presents different biomedical signal processing challenges each year. Challenges include detection of sleep apnea[8] using only the ECG and developing "a fully automated

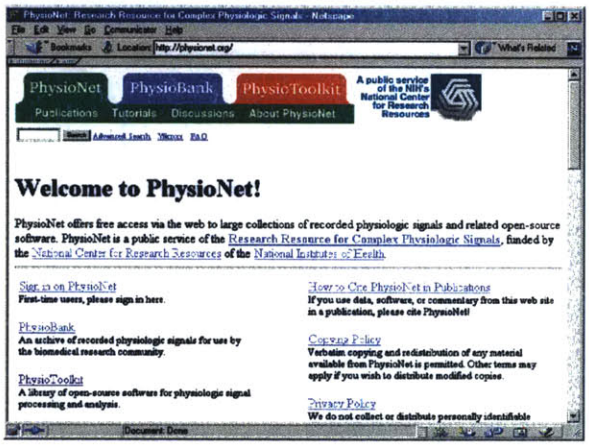


Figure 4 *PhysioNet* webpage.

[7] See http://www.cinc.org for more information.
[8] "Detecting and quantifying sleep apnea…"

method to predict the onset of paroxysmal atrial fibrillation/flutter(PAF), based on the ECG prior to the event."[9] In addition to supporting an on-line community, PhysioNet offers educational tutorials to assist investigators in effectively using the Resource. PhysioNet is located at http://www.physionet.org. Figure 4 is a snapshot of the PhysioNet webpage.

2.2 WAVE – A signal and annotation editor

Within PhysioToolkit, *WAVE*[10] is an existing tool built for clinicians and researchers. It is designed to display and manipulate physiological signal data and annotation data found in PhysioBank. *WAVE* includes data visualization functionality, search by time frame and/or annotation capability, and "chart recording"-type[11] display. In addition to data viewing, *WAVE* also allows researchers to view and create annotation files for signal data records. For those who wish to run different processing algorithms on the signals, *WAVE* also provides an interface for exterior data analyses tools. Lastly, Wave can read data and annotations from local files as a remote HTTP (web)[12] or FTP[13] server, thus, functioning as a special purpose web browser. *WAVE* is an application written in C++ that must be downloaded before execution on a user's machine. Any physiologic data found in PhysioBank can be downloaded and viewed using WAVE. This application was originally designed for SunOS or SPARC-based systems using the open-source XView toolkit.[14] Several years ago, Wave was ported to Linux-based PCs. As of July 2001, a

---

[9] "Predicting paroxysmal atrial fibrillation…"
[10] Moody, GB.
[11] "Chart recording" displays refer to traditional physiologic data displays used by medical personnel.
[12] HTTP: Hyper-text Transfer Protocol
[13] FTP: File Transfer Protocol

[14] XView toolkit: An object-oriented library that enables objects such as windows, text, panels, icons to construct an application.

new Gimp Tool Kit (GTK +) version of WAVE called GTKWAVE has also been released. This version supports Linux/Unix and MS-Windows platforms. Figure 5 shows the WAVE interface display record 100 from the MIT-Beth Israel Hospital Arrhythmia database.

Figure 5 WAVE Screenshot with Record 100 from MITDB.

3 Requirements and Specifications

3.1 Overview

This thesis project provides a new tool within the *WAVE* family of applications that can be used via a web browser. This new software tool, nicknamed "*Java WAVE*", provides signal and annotation viewing capabilities similar to some basic *WAVE* features, but exists as a Java applet[15] that can be run within a web browser. "*Java WAVE*" is a solution for users who wish to browse through records and/or view records in *PhysioBank* without manually downloading any data or applications as in recent versions of the original WAVE application. As an applet, this tool utilizes the computation power

[15] Applet: A program written in Java that can be included in an HTML page similar to the way an image is included. (See http://java.sun.com/applets/index.html for more information.)

of a client machine with minimal strain on the server machine. The use of an applet also allows the program to be initialized by the simple click of a URL (universal record locator) within popular web browsers such as Netscape. The use of Java makes this program portable (see Section 4.1.1) and begins new investigations of re-implementing *WAVE* as an object-oriented application.

3.2 Applet Specifications

The following segment describes the program's requirements. These specifications are broken down into two sections. Section 3.2.1 will describe all of the graphical user interface (GUI) requirements of this program. This refers to all features that are visually displayed and/or can be manipulated by the user. The section following describes the underlying components that contribute to the retrieval and storage of record and annotation data. This section describes requirements for the applet and *not* the actual applet.

3.2.1 Graphical User Interface (GUI) requirements

The GUI must contain the following features:

- *Scale grid* – A grid that measures 0.5 mV/0.2 sec. This tool gives users a reference for scale measurements. The grid can be toggled on and off.
- *Time display* – Both start time and end time of the signal segment must be shown in the screen. Time display is used to display the location of the record shown. Both displays must be shown in *hh:mm:ss* format (*h* = hours, *m* = minutes, *s* = seconds.
- *Signal display* – The GUI must display *all* signals within a record accurately.
- *Signal name display* – Likewise, the GUI must identify each signal and display the name of the signal correctly. This feature can be toggled on and off.

- *Annotation display* – Any annotations linked to a record must be labeled properly within the record displayed. This display uses green lines to allow better visualization of annotations. The annotation feature can also be toggled on and off.
- *"Go to" Time* – The GUI must provide a way to "go to" a user-specified time within the record. This function must be able to search the record in both directions (both forward and backward.) Appropriate error checking for user inputs also must exist
- *Search Annotations* – Analogous to the time "goto" function, this GUI must have appropriate methods for allowing a user to search for specific annotations. This search can also search both forward and backward.
- *Time zoom* – This function allows the user to change the time scale among five different settings. Changing the time scale will shift the focus of the screen from a couple of seconds to, potentially, the entire record.
- *Signal amplitude zoom* – This feature changes the amplitude of the signals shown on screen. Like the time scale, there must be at least five different settings.
- *Navigation tools* – The final requirement is navigation. This function is extremely important because it allows the user to navigate through the entire record. The applet must support both forward and backward navigation methods.

3.2.2 Signal and Annotation Storage

Underneath the GUI, this applet engine must be able to do the following:

- *Record reading* – The applet must read the given input record data from a web page. Currently, this data is spawned from a cgi-script[16] called *rdsamp*[17] and served as a web page of text data.

[16] CGI script: Common Gateway Interface scripts allow execution of scripts on the host to create dynamically generated information for the client.
[17] *Rdsamp* can be found in WFDB tools at http://www.physionet.org.

- *Annotation reading* – Likewise, annotation reading from a webpage is necessary in order for the applet to pull annotation data. This data is found in text format on a web page using a cgi-script called rdann.[18]
- *Record storage* – After reading the data, the applet must be able to store a complete record or simulate the storage of a complete record. If the storage is simulated, the design should be modular enough such that GUI portion of the applet does not need to be adjusted to compensate.
- *Annotation storage* – The annotation data must also be stored within the applet. Again, this data can either be fully stored or simulated as fully stored. The GUI portion of the applet should not be affected if the full data storage is simulated.
- *URL input* – Implicit to record and annotation reading is the capability to take two different variable inputs for records and for annotations. This variable input must be in the form of URLs (Universal Record Locators). These URLs contain the data pages spawned by *rdsamp* and *rdann*.

3.3 User's Manual

This is the user's manual for the signal and annotation viewer Java applet. This manual does not contain information for server side installation.

3.3.1 Loading the Applet

To load the applet, first open a Netscape browser. Netscape is the sole supported browser for the Java WAVE applet. After opening up the browser, initialize the applet by clicking on the appropriate URL that specifies the desired record or desired record/annotation combo to display. Figure 6 shows an example of a list of record links.

[18] *Rdann* can be found in the WFDB tools at http://www.physionet.org.

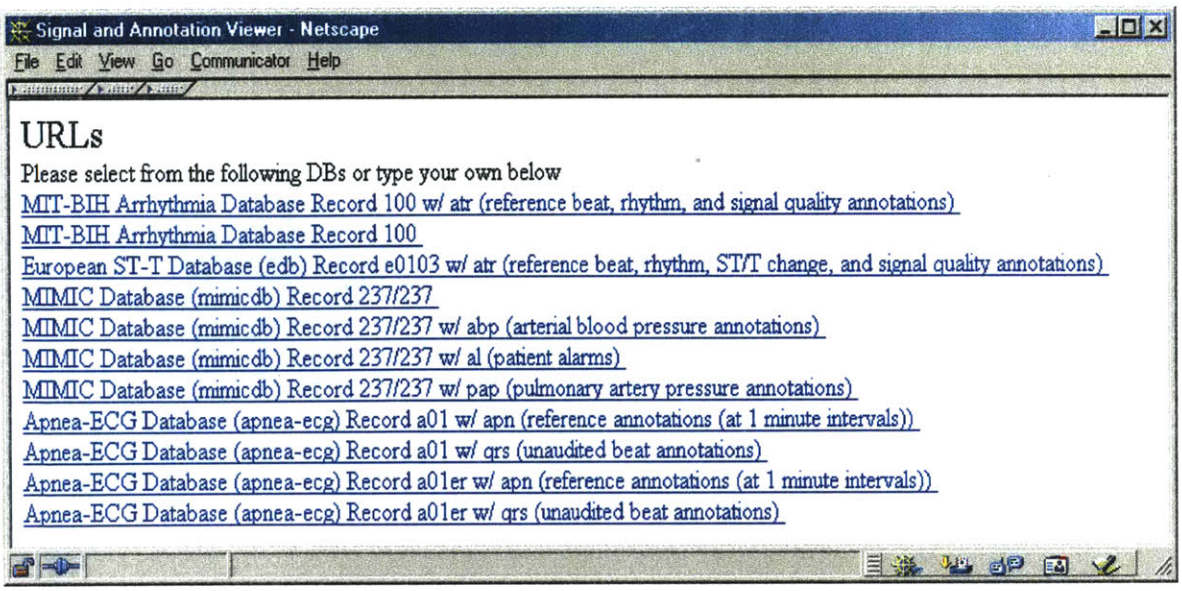


Figure 6 List of URLs specifying record/annotation combinations

Alternatively, one can also enter the record and annotation information through some input fields. When loading the applet for the first time, the user may be prompted to download a Java Swing plug-in. This plug-in is necessary in order to run the applet. The applet will be loaded and displayed in a new web page after this process is complete.

3.3.2 Navigation Control

The navigation controls are found at the bottom left hand corner of the applet. Figure 7 shows precisely where these controls are located.

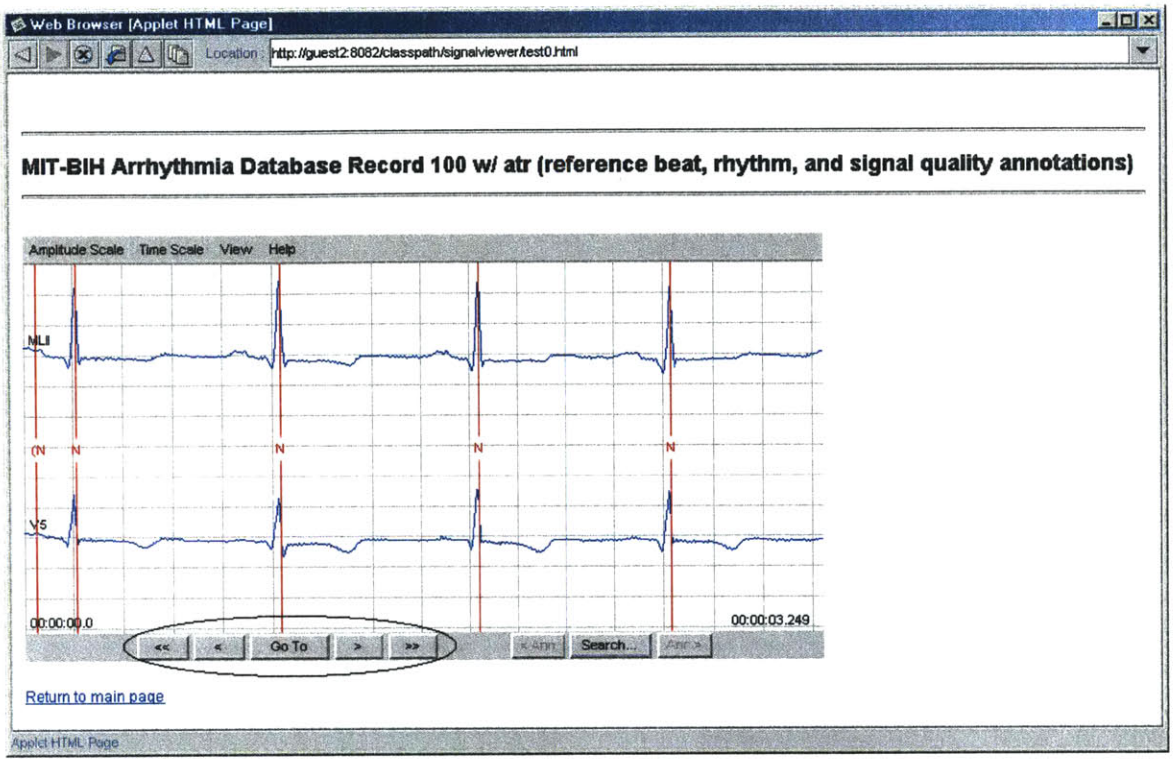


Figure 7 Applet window with navigation controls circled

3.3.2.1 Scrolling through a Record

To scroll forward by half a screen, click the > button. To scroll forward by a full screen, click the >> button. Likewise, to scroll backward, click the < button to move half a screen, and the << button to move a full screen.

3.3.2.2 Finding a Specific Time

In order to find a specified time, first click the "Go To" button. This click prompts dialog box which pops up in the middle of the applet screen. Figure 8 is a snapshot of the dialog box. The dialog is set by default

Figure 8 "Go To" dialog box

to 00:00:00. Enter the desired time in the form of hh:mm:ss and press OK. If the time is entered incorrectly, a time format warning dialog will appear. If the time is not found within the applet, a "Time Does Not Exist within Record" dialog box will appear. Otherwise, the applet will find the time and display the record starting from that time. If the "Time go to" button is clicked unintentionally, click the Cancel button.

3.3.3 Toggles

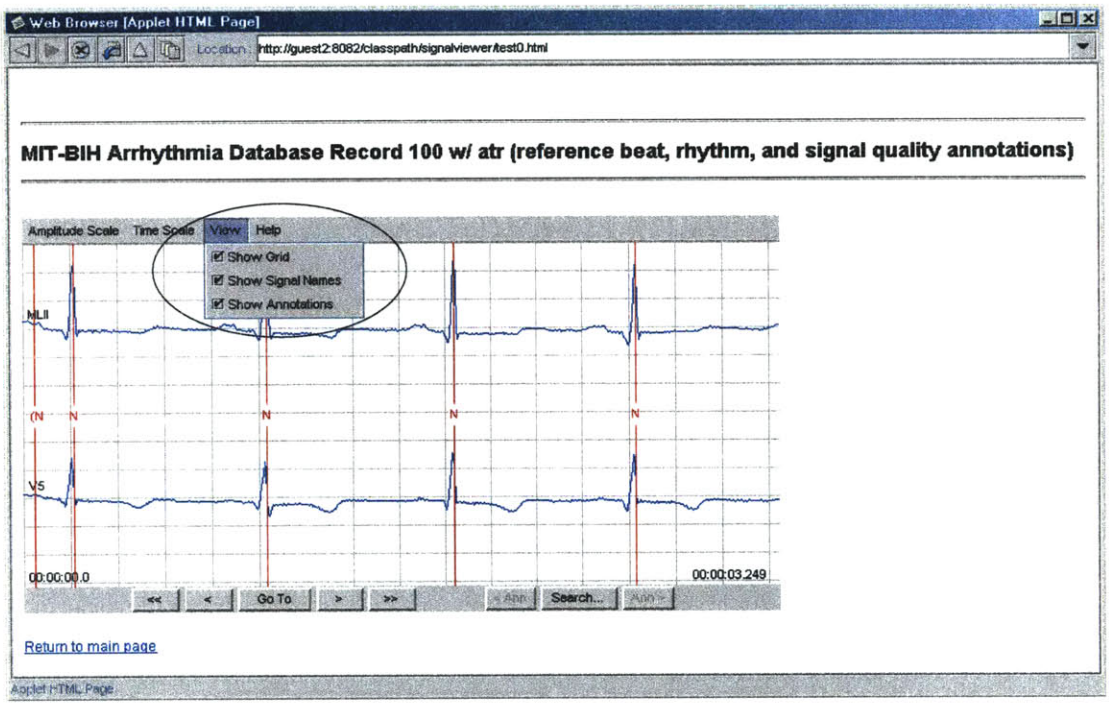


Figure 9 Applet window with toggle options circled

The following properties can be toggled on and off: show grid, show signal names, and show annotations. Figure 9 displays the location of the toggle options menu.

The grid will be displayed upon applet loading. Each vertical line in the grid is spaced 0.2s apart, and each horizontal line in the grid is 0.5 mV apart. To turn off the grid, go to the View menu and open. There should be a check next to "show grid." Click this box

and the grid will be turned off. To turn on the grid, simply go back to the menu item and click again.

The signal names are also displayed when the applet is loaded. In order to toggle off the signal names, go to the View menu again. Click the box next to "show signal names" and the checkmark will disappear and the signal names will no longer be displayed. The last toggle button turns the annotations on and off. If the applet is loaded with an annotation file, the annotations will be displayed. To turn off the annotation, go to the View Menu and select "Show Annotations." After selecting this item, the annotations will be turned off and the annotation search buttons will be disabled. To redisplay the annotations and re-enable the search buttons, click "Show Annotations" again. If the applet is not initialized with an annotation file, the "Show Annotations" item in the View menu will not be enabled.

3.3.4 Search By Annotation

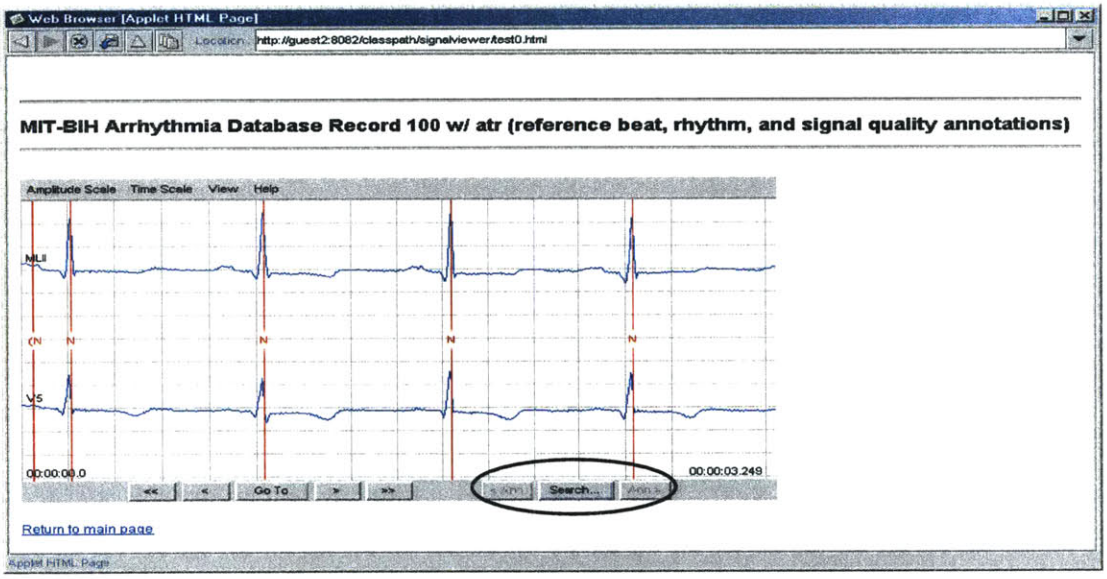


Figure 10 Applet window with annotation search buttons circled

The search annotation functions are only enabled when the applet is loaded with an annotation file *and* when the "Show Annotations" menu item is checked. These buttons are found in the lower right hand corner. Figure 10 illustrates their location. Before the first annotation search the "< Ann " and "Ann >" search buttons will be disabled. To search for an annotation, click "Search Annotation." Clicking this button prompts a dialog box that requests an annotation to search. Figure 11 is a picture of this dialog box. The default annotation to search is "A." Enter the desired annotation and press OK. A new dialog box will appear stating the annotation to be searched. Press OK to dismiss the dialog. The forward and backward search buttons are enabled at this point. Figure 12 shows the enabled search buttons. To search forward in the record for an annotation, press the "Ann >" button. If that annotation exists within the rest of the record, the applet will be redrawn to display the annotation found in the middle of the canvas.

Figure 11 Annotation search dialog box

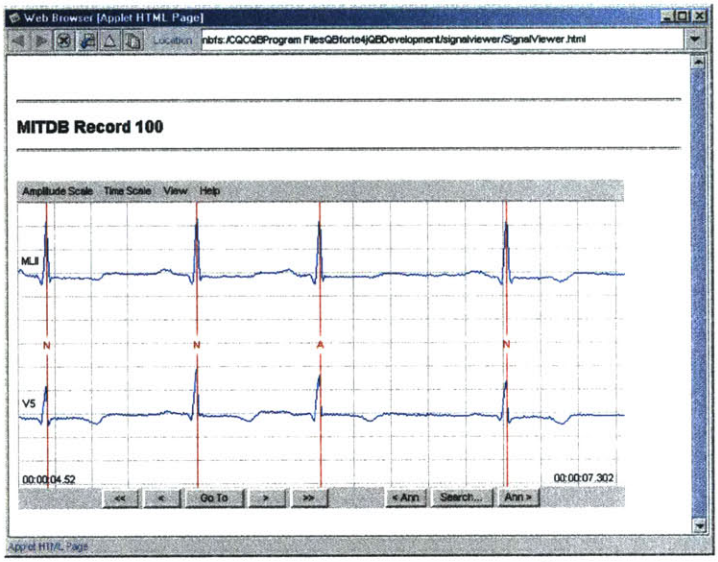


Figure 12 Annotation Found

If the annotation does not exist in the rest of the record, an "Annotation Does Not Exist" dialog warning will pop up. To search backward in the record for an annotation, press the "< Ann " button.

The "< Ann" button behaves exactly like the "Ann >" button, but searches the record backward. If the "Search Annotation" button is clicked by accident, simply click the "Cancel" button and the dialog box will disappear from the screen.

3.3.5 Magnification

The user may desire to either focus on a specific detail of a given signal or desire to see the bigger picture of the record. Time and amplitude magnification check boxes are provided to support zooming in and zooming out. Figure 13 pinpoints the location of these check boxes. The grid will change size whenever the magnification factors change, because the units per gridline remain constant.

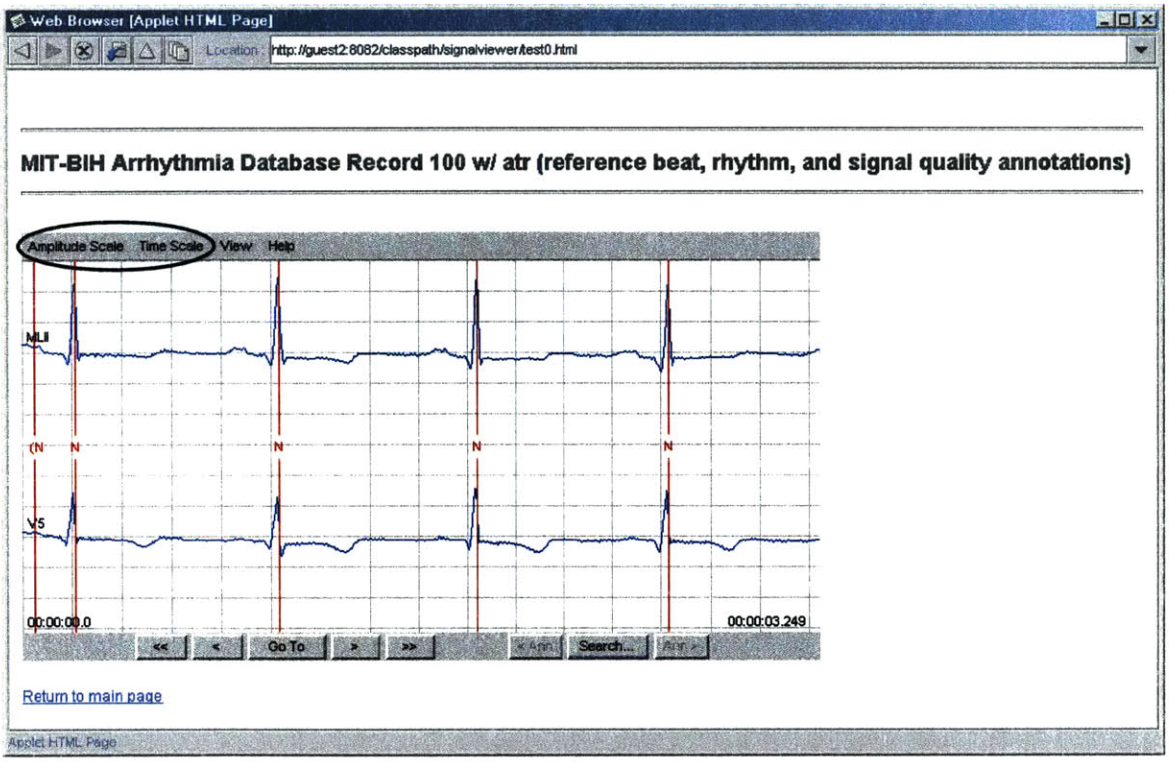


Figure 13 Applet window with Time and Amplitude Scale menus circled

3.3.5.1 Time Scaling

In order to alter the time scaling, select one of the check box options under the "Time Scale" menu in the menu bar. Five possible time scale options exist.

3.3.5.2 Amplitude Scaling

In order to alter the amplitude scaling, select one of the check box options under the "Amplitude Scale" menu in the menu bar. Five possible amplitude scale options exist.

3.3.6 Exiting the Applet

Redirecting the browser to a different URL will remove the applet from the browser window.

3.4 Performance

This software system requires a Java Virtual Machine (JVM)[19] and sufficient physiologic data from *PhysioBank* for normal operation. The JVM must be able to run bytecodes produced from the Java 2 platform with the Java Foundation Classes (nicknamed *Swing*). Any Java technology-enabled browser will have a JVM. Browsers without JVMs that can run *Swing* require a downloadable browser plug-in.[20] The physiologic data is expected to be provided via URLs that use *rdsamp* or *rdann* to retrieve data in text format.

From a time analysis perspective, most of the applet's major functions run in linear time. Loading the record and annotation data runs in O(n) where n is the length of the record or annotation. The signal drawing process also takes O(n) where n is the number

[19] Java Virtual Machine(JVM): A software implementation of a "CPU" used to run compiled Java code.
[20] Plug-in: Programs that can be easily installed and used as part of a Web browser.

of sample points displayed on a single screen. Although the process is linear, this is the best asymptotic behavior possible because all the sample points displayed must be processed. Like the signal drawing process, the annotation drawing process takes O(n) where n is the number of annotations displayed on a single screen. Searching through the annotations also takes O(n) time. In this case, n is the number of all annotations within a file for the record displayed. O(n) is the time boundary because the worst case scenario would be a forward search at the beginning of a record (or a backward search at the end of a record) where the annotation type searched for does not exist. The final major function is the only exception to being linear. Going to specific times runs in constant time or O(1). This is because the time specified is multiplied by the frequency of the sampling to find a start sample that is used to initialize a redrawing to the screen.

From a space analysis standpoint, the overall software system takes O(n) space. This results from the following equation:

$$O(n) = O(n) + O(n)$$

The first O(n) within the equation represents the space taken by the record data, thus n is the length of the record. The second O(n) represents the annotation data where n is the length of the annotation. All other functions use constant space other than the space used by data from the existing record and annotation objects.

4 Design

The heart of the program is the design. In order to create a software system that fits the outlined requirements and specifications and also remains useable and viable, it is important that the system follow certain theoretical criteria. The following section will

detail these criteria and explain certain implementation issues and strategies in terms of these criteria.

4.1 Criteria

The following six criteria provided the basis for this project: portability, intuitive usage, efficiency, modularity, extensibility, and scalability. The first three refer to program specific goals and the last three refer to general software engineering goals.

4.1.1 Portable

Portability describes a software application that can be executed on any computer platform. In order to maximize the usability of this thesis project, it is required that this applet has the ability to run within some browser on the following three platforms: Linux/UNIX, MacOS, and MS-Windows. Ensuring that the signal and annotation viewer runs on these platforms provides the users of these platforms a viable solution to easily browse through data within *PhysioBank.* Portability also ensures that minimal user intervention (at most, downloading the correct browser plug-in) is necessary to run this applet.

4.1.2 Intuitive

Intuitive usage refers to the property that a program is easy to use. This tool must be simple enough for a user to initialize without difficulties. It also must have a shallow learning curve. In contrast to the original WAVE program, "Java WAVE" should attempt to reduce the number of user interventions and require neither a software download nor installation. In addition to ease of start-up usage, the software also must

have an intuitive, user-friendly interface. When a user begins to use the applet, he or she should be able to navigate and manipulate the visual output without training.

4.1.3 Time efficient

In order to enhance this project's usefulness as a browser for signal and annotation data, the applet must be both quick to load and quick to display. If a user has to wait too long for one record to load and display, the effectiveness of the applet will diminish because the user will likely look for a faster alternative or decide not to preview the data at all. Most users surfing the Internet will flip to a different web page if the current page has not loaded within four or five seconds. Since this applet will also be displayed via webpage, it is necessary that the load and display times occur within 4 seconds.

4.1.4 Modular

Modularity is described in Merriam-Webster's collegiate dictionary as "constructed with standardized units or dimensions for flexibility and variety in use." One way to achieve modularity is through the use of an object-oriented programming system. Modularity is particularly useful in a few respects. The first is that modular construction allows different parts of the program to be "black boxed." When a part or object of the program is "black-boxed," any component that interacts with this object will not need to understand the interior workings of the object. Other components only need to recognize that the object guarantees to fulfill its specified functions and provides unchangeable methods to access these functions. A modular system will also ensure that when one part of the program "breaks," the rest of the code will not break. Other useful

aspects of modularity are that modular programs are easier to debug because problems will be easier to isolate and that they are easier to extend because the program is in distinct parts.

4.1.5 Extensible

Extensibility refers to how well a program can be modified when features are changed or added. An extensible program is important because the specifications may change frequently depending on client and user input. For this applet, extensibility is necessary because researchers often desire more features that allow them to study waveforms more carefully. Some potential additions and/or changes are re-implementing data reading techniques to increase speed and adding more GUI functionality. In addition, this applet must have the capability to support further development such that a complete Java version of *WAVE* can be implemented using this framework.

4.1.6 Scalable

Scalability is the robustness of a software system when forced to deal with larger inputs. A scalable program will not fail under stress and will still perform reasonably. If this signal and annotation applet is scalable, it will not break when given a large amount of data such as a 48 hour record with a three millisecond sampling frequency or a record with twenty signal waveforms. In addition to not breaking, the applet program must be able to load and display this data without long delays.

4.2 Implementation

Due to the nature of this project, there are many different implementation solutions. The choice of different implementation tools becomes extremely significant in

satisfying the above design constraints. One choice is a server-side solution in which the server processes the signal data and produces a raster image[21] that is sent to the requesting client computer. An existing solution within *PhysioToolkit* called the *Chart-O-Matic*[22] uses this model and sends a complete PNG[23] image to the client. Figure 14 is an image of the *Chart-O-Matic.*

Figure 14 Chart-O-Matic

A second possible implementation method for this program is the use of SVG or Scalable Vector Graphics, a vector graphics language written in XML.[24] This method has the advantage of reducing network traffic, because SVGs are transmitted as a vector rather than a two-dimensional array which reduces file size. SVGs are also resolution independent and scale down or up to proportionally fit any size display on any net device.[25] Unfortunately, the use of SVGs is still very new and web support for SVGs is still being developed. A third solution is to use the existing WAVE as a web browser, as mentioned earlier. The final implementation tool considered was using a Java applet.

The first three design criteria state that this software system must be portable, intuitive, and efficient. Implementing this project using a Java applet implies the

[21] Raster: The rectangular area of a display screen actually being used to display images.
[22] Chart-O-Matic can be found at http://www.physionet.org/cgi-bin/chart
[23] PNG: Portable Network Graphics, a new bit-mapped graphics format similar to GIFs.
[24] XML: Extensible Markup Language, a language used for web documents.
[25] http://www.pcwebopedia.com/TERM/S/SVG.html

fulfillment of the first criteria. Using Java also provides the means to achieve intuitiveness and an acceptable level of efficiency. This software system was developed using Sun Microsystem's community edition of Forte for Java™, an integrated developing environment (IDE) under Sun Microsystem's version of Java, the Java 2 Standard Edition Platform version 1.3.1.[26] See Figure 15 for a look at Forte for Java™.

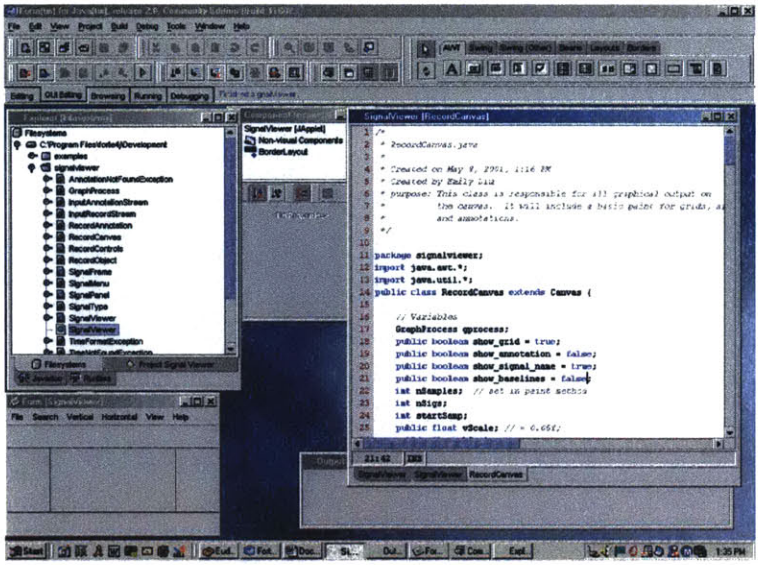


Figure 15 Forte for Java IDE

4.2.1 Java Platform

Java is an object-oriented programming language that generates a genericcompiler "byte" code. This "byte" code can be that run on the client side using the Java Virtual Machine that exists on the user's machine. For this reason, Java is extremely portable because it can feasibly run on all platforms and perform exacty the same way across all platforms according to Java's goal and motto, "*write once, run anywhere*."[27] This Java advantage also translates to the same advantage for this signal and annotation tool, thus, satisfying the requirement of portability. A Java-based program will run similarly on major platforms including UNIX/Linux, MacOS, and MS-Windows.

[26] Java 2 version 1.3.1 can be downloaded from http://java.sun.com/j2se/1.3/
[27] See http://java.sun.com for more information.

Using Java also allows this program to run as an applet. As defined in a previous section, applets are embedded in a webpage and have the ability to run in all popular browsers. This gives users the ability to execute applets without installation when proper applet capabilities exist (the browser's JVM needs to run the appropriate version of Java) simply because they're initialized when the webpage is opened. Since an applet circumvents the need to manually download and install, this implementation of a new WAVE-like tool partially fulfills the requirement of intuitive usage.

Another benefit of using an applet is the ability to utilize the power of client-side computation. This architecture provides faster delivery of data by lightening the server load, since data processing is moved to the client computer. Client-side computation also provides the greater benefit of reducing the amount of information transmitted through the network. Instead of processing an image on the server side and sending data-heavy raster images (GIFs and PNGs) to the client, the server only needs to transmit signal data (vector graphics) to the client. The bottleneck caused by limited network bandwidth can be relieved using this model. Thus, the software system is more efficient because data load time is shortened.

Finally, Java's object-oriented nature makes it a good choice for developers who wish to program modularly. Java forces developers to program using different structures called classes which can encapsulate the intuitive definition of an object by organizing all methods pertaining to the object in the same section.

4.2.2 Java Foundation Classes - *Swing*

A key reason for specifically choosing the Java 2 platform versus an earlier version of Java is the addition of the Java Foundation Classes, nicknamed *Swing*. *Swing*

builds on Java's existing GUI capabilities, the Abstract Windows Toolkit (AWT)[28], and provides Java developers more flexibility in developing GUIs. New "lightweight" classes, classes that depend entirely on Java instead of the native platform, are easier to control, more robust, and behave more similarly on different platforms. *Swing's* GUI capabilities also exceed that of AWT and provide more comprehensive GUI abilities for actual product development. In addition to these structural changes, *Swing* also has a cosmetic "Look and Feel" feature that allows developers to customize the GUI according to platform in order to present a familiar user interface to those who use the applet for the first time.

4.3 Module Structure

Although a Java implementation of the software system satisfies many of the design criteria, the last three criteria – modularity, scalability, and extensibility – are primarily fulfilled through design decisions pertaining to the actual code. This section outlines the structure of the system and explains the design decisions behind it. The code is broken up into three major sections – the GUI, the data structures, and the data reading objects – and a fourth component that ties the GUI and the data together.

Much of the impetus behind the design was to preserve modularity and thus, extensibility. On the highest design level, objects pertaining to the data (including reading and writing) and the GUI are separated so that changes in the GUI modules will not affect data modules and vice versa. With this separation, if the data structure or data storage methods are changed or extended, the GUI objects will not break as long as it is given data in the form it expects. The data modules also need no information about the

[28] For more information on AWT, refer to http://java.sun.com

way data is displayed, but must only guarantee to provide the proper methods to access data.

At the second level of design, the data objects are separated into data structures and data reading objects. This decision gives data reading methods flexibility without affecting the data structures. In this manner, the data reading methods can now be rewritten as threads[29] and/or changed to read from different types of inputs.

The last component of the design connects the GUI and the data structures together. This module was created to preserve the separation between the GUI and the data.

4.3.1 GUI Modules

The GUI is broken up into three different modules – the applet, the canvas, and the control panel. This modularization allows changes to either the canvas or the control panel without affecting the other. When the canvas' paint (print to screen) methods are altered, the control panel module will still function appropriately. The applet module contains these two components. The applet, canvas, and control panel are interdependent.

The following modules constitute the GUI portion of the applet:

- *SignalViewer* – This is the highest level GUI container. SignalViewer inherits from the javax.swing.Japplet class[30] and has all the functionality of a Java swing applet. This object is the executable class for the system and contains the menu bar which controls features including magnification and toggle functionality. In addition to initializing the program, SignalViewer initializes RecordCanvas, RecordControls, and GraphProcess.

---

[29] Threads: Placeholder information associated with a single use of a program that can handle multiple concurrent users.
[30] javax.swing.Japplet can be found in the Java 2 Platform API at http://java.sun.com/j2se/1.3/docs/api/index.html

- *RecordCanvas* – RecordCanvas is the drawing canvas of the applet and inherits from the java.awt.Canvas class.[31] It is responsible for drawing all objects to the screen including the grid, signal names, time display, signals and annotations. RecordCanvas also holds several state variables of the system including Boolean variables that indicate which objects should be drawn and an integer variable that indicates the starting sample point of the screen. RecordCanvas uses GraphProcess to gather all data that needs to be displayed and accesses SignalViewer to gather initializing variables.
- *RecordControl* – This object is inherited from the javax.swing.JPanel class[32] and is the control panel displayed within the applet. It contains the action buttons that affect RecordCanvas via the higher level SignalViewer class. These buttons are arranged in two clusters. The first cluster features a time "go to" button and four other forward and backward navigation buttons. The second cluster is only displayed when annotations are loaded with the record. These buttons have forward and backward annotation search capabilities. Whereas RecordCanvas prints everything to screen, control panel manipulates the printed output and controls what is to be displayed.

4.3.2 Data Structures

The data structures are separated by definition. A different structure was created for records, signals, and annotations. Each structure stores all the properties and the data concerning the object represented.

These data structures hold all the data including signals, records, and annotations.

- *RecordObject* – The RecordObject is a data structure that contains all the data represented by a record within *PhysioBank*. It holds the record name, the sampling frequency, and a list of all the signals within the record. For software scalability, RecordObject keeps an expandable list of SignalTypes rather than actual signal data. This design strategy provides for a variable number of signals

[31] The API for java.awt.Canvas can also be found at http://java.sun.com/j2se/1.3/docs/api/index.html
[32] javax.swing.JPanel's API is found at http://java.sun.com/j2se/1.3/docs/api/index.html

per record. Although RecordObject does not actually contain the data for each signal, it provides standard methods for read/write access to all the data within the record including signals. These standard methods allow RecordObject to hide its implementation details from other objects that are sending and requesting data (Black box methodology). These hidden details provide for a more robust system. RecordObject initializes data reading through InputRecordStream (Section 4.3.3).

- *RecordAnnotation* – This data structure contains all data pertaining to one annotation file. Like RecordObject, it provides a host of methods that access the data within the file. Thus, RecordAnnotation's implementation is also hidden from other objects in order to be scalable and extensible. RecordAnnotation initializes data reading through InputAnnotationStream (Section 4.3.3).
- *SignalType* – SignalType contains all the data concerning one signal. It holds the signal name and all signal data points. Although SignalType currently contains all the signal data for an entire record, its interface is designed to allow implementation as a virtual array which will permit different strategies concerning data storage. This virtual array is an extensible implementation that will always appear to have all the signal data through provided methods although only part of the data may exist in storage. This design implementation will ensure that other modules do not break if the signal object storage changes and it will also provide "hooks" to make the system more scalable. Currently, the applet cannot store records with too many data points (48-hour records or records with 20 signals,) because the browser which is running the applet will crash when the browser's memory limit is exceeded.[33] In order to facilitate a virtual array, SignalType contains information about the signal length (number of samples) and the sample start point.

4.3.3 Data Reading Objects

The following objects read data from a URL input stream and store it in the data objects.

---

[33] Browsers are allocated a finite amount of memory by the host computer.

- *InputRecordStream* – This object reads data from a URL and stores it into RecordObject and SignalType using their access methods. InputRecordStream takes a text input representing a URL. This URL accesses a cgi-script in the *PhysioToolkit* called rdsamp that serves record data as text output. The record and record length information is specified using rdsamp. (More information about the format of rdsamp is found in Appendix A.) Changing the data reading implementation only requires changing this class.
- *InputAnnotationStream* – InputAnnotationStream reads annotation data from an input stream and stores it into RecordAnnotation. Like InputRecordStream, InputAnnotationStream reads data from a URL that accesses a cgi-script called rdann which converts data from *PhysioBank* into a text web page (see Appendix A for rdann format). Changing the annotation data reading implementation also only requires changing this class.

4.3.4 GraphProcess

GraphProcess is a component that does not fit into any of the three major groups, but is one of the most important structures because it links the GUI modules to the data structures. This object gathers raw data by accessing RecordObject and RecordAnnotation and processes the data before giving it to RecordCanvas. Then, RecordCanvas takes this data and paints it to the applet canvas. With this implementation, RecordCanvas and the rest of the GUI components never need to know anything about the data objects. Likewise, this design ensures that the integrity of the data is preserved.

A key display algorithm to note is a variation on the Turning Point Compression (TPC) algorithm. The TPC algorithm is used to compress the data points when there are more sample points than horizontal pixel points on the screen. This variation basically maps a certain number of sample points to a particular pixel and displays a sample point

by choosing either the local maximum or minimum within the group of sample points. The local maximum or minimum is chosen based on a comparison between the first sample point in the group, variable y, and the last sample point, variable x, which was the sample point chosen for the previous pixel. For the base case, the first sample point is plotted. If sample y is less than sample x, then choose the local minimum in the group. If sample y is greater than or equal to sample x, then choose the local maximum of the group of sample points. The Turning Point compression algorithm is a simple and common ECG compression algorithm used in biomedical circles because it preserves key characteristics of signals.

Other than display processing, GraphProcess also contains a searching algorithm for annotations. The annotation search is a straight linear search through the list of annotations.

5 Testing

The testing strategy of this software system was black-box testing. Each module was tested for functionality only through the visual display. The overall system was tested for stress load and portability.

The GUI component was tested by user interaction and visual checks. A combination of user and visual testing ensured that all the buttons on the applet worked as designed. All the menu features including time and amplitude magnification, different toggle abilities, and the information dialogs were also user tested and visually assured. These tests included checks for accurate visual display (correct waveforms, grid spacing) and functionality tests. All the GUI components work as intended.

The data structures and the data reading components were checked using print methods created for testing. Using these methods, the stored data is checked against the URL text input to guarantee accuracy. These print methods exist for each data structure. The data structures and data reading modules also work correctly.

In order to test the scalability of the system, the applet was loaded with different types of records and annotation files within *PhysioBank* for stress testing. This test showed that the applet works well with any records of the same format as the MITBIH Arrhythmia database (the original test record). Records that have a signal with more than one word within its name are currently unreadable. However, records with a large number of signals (greater than five) can still be displayed accurately. Currently, all annotation files that display text annotations are also displayed correctly.

The last check of the system was platform testing. This test determined that the applet could run accurately using at least a Netscape 4.7 browser on the Linux/Unix, MacOS, and MS-Windows platforms provided that the browser downloaded the correct Java Swing plug-in.

6 Reflection

6.1 Evaluation

The major success of this project is the working applet. However, keeping the software structure modular proved to be a key component to gaining that success. Once the structure was decided, the flexibility from modularization allowed a variety of changes to occur without rewriting the rest of the code. The separation between the GUI and the data objects was invaluable because the GUI changed frequently to display the many variations of data. Since the data reading methods were implemented several

different times before achieving steady state, dividing the data structures and the data reading devices was very helpful. Most additional features, such as selective signal display and more scaling options, were also easily added to the applet because the methods already existed to process the data.

Although the applet was successful in fulfilling its specifications, a mistake within the software design occurred due to the lack of foresight concerning record and annotation formatting. Each database within *PhysioBank* has its own header file which describes properties of the database and specific formats used within the records of that database. This system was developed without considering that knowledge and has no way to access that header file and thus, no methods to determine the structure of the input file.

There are two major performance problems with respect to storing records, all of which stems from the same mistake. The first is that the software currently does not support records or annotations with formats varying from the test record and annotation. This is because the applet does not properly access a header file for each database which describes the format of the data. Another limitation from this same mistake is that the signals may not be calibrated correctly. Since this program does not use *wfdbcal*[34], a tool used by the orginal WAVE to calibrate waveforms, the signals based on the header file information, some signals, such as ventricular blood pressure, do not scale correctly on screen. The last performance problem is the limitation that the applet cannot display long records with too much data, as mentioned previously. Addressing these issues is the highest priority of the planned follow-up research.

---

[34] wfdbcal can be found in PhysioToolkit at http://www.physionet/org/physiotools

6.2 Key Lessons

The key lessons of this project can be grouped in three sections: the technology, a flexible design, and a good process.

First, using a changing technology is a risk. The changing technology of Java and specifically Swing made it difficult to use. In spite of the rapidly shifting nature of technology, Swing, Java's newest GUI toolkit, seemed like an ideal choice for building a flexible GUI. However, the downside of using a new technology was a steep learning curve because there were few contacts for help from other Swing developers. In addition, Swing was not yet a fully supported popular tool. At the time of project proposal, the GUI developing community was moving toward using Swing, but the toolkit was not supported by the popular browser from Microsoft, Internet Explorer(IE). At the time of project completion, Swing has yet to be supported by IE.

The second lesson is that an established flexible design *before coding* is extremely important. At the beginning of the project it was very easy to keep rewriting the same modules repeatedly in order to account for all the features. However, the project was always evolving, so using that method became inefficient and time consuming. After a design was established, its flexible nature proved very useful, because fulfilling new specifications that were added mid-project did not require reworking much of the previously written code.

The final lesson was purely an organizational lesson on process. Well outlined specific project goals were important in understanding the amount of work to be done. In addition to specifying the goals, it was also essential to make a timeline with key dates

for the achievement of certain milestones. These two key elements provided a good framework for understand the problem more clearly and completing the project.

7 Conclusion

Where WAVE provides researchers a wide range of tools to study and manipulate signal waveforms and annotation files, this new Java signal and annotation tool addresses the different goal of giving researchers and curious users the ability to browse through records and annotations files on *PhysioBank*. Using Java applets and certain design constraints including modularity, scalability, and extensibility, this software system has achieved its goal of providing a fast, portable, and easy to use tool that utilizes client-side computation through a web browser.

Follow-up research will occur immediately to correct some of the current limitations of this new applet tool in order to post it on *PhysioToolkit* as soon as possible. As mentioned before, the data reading modules will be extended as threads so that only the section of the data being viewed must be stored in the applet at one time. A further optimization includes changing the data reading methods to read from the actual binary files stored on *PhysioBank* rather than using rdsamp or rdann to serve the data. This change will cut down on server side processing and hopefully provides faster data loading. Overall, this existing research is only the beginning of a project to achieve a full-scale Java implementation of WAVE.

Bibliography

"Detecting and quantifying sleep apnea based on the ECG: A challenge from PhysioNet and Computers in Cardiology 2000." 8 February 2001. http://physionet.org/cinc-challenge-2000.shtml (3 Aug. 2001).

Goldberger AL, Amaral LAN, Glass L, Hausdorff JM, Ivanov PCh, Mark RG, Mietus JE, Moody GB, Peng CK, Stanley HE. "PhysioBank, PhysioToolkit, and Physionet: Components of a New Research Resource for Complex Physiologic Signals." *Circulation* 101(23):e215-e220 [Circulation Electronic Pages; http://circ.ahajournals.org/cgi/content/full/101/23/e215]; 2000 (June 13).

Moody, GB. *Wave User's Guide*. http://www.physionet.org/Physiotools/wave. Massachusetts Institute of Technology, 1992-1997.

Moody GB, Mark, RG. "The Impact of the MIT-BIH Arrhythmia Database." IEEE Engineering in Medicine and Biology Magazine, Volume 20, Number 3, May/June 2001, pp.45-50.

Moody GB, Mark RG. "The MIT-BIH Arrhythmia Database on CD-ROM and software to use with it." Computers in Cardiology 1990. IEEE Computer Society Press, Los Alamitos, CA, pp. 185-188

.

Moody GB, Mark RG, Goldberger AL. "PhysioNet: A Web-Based Resource for the Study of Physiologic Signals." IEEE Engineering in Medicine and Biology Magazine, Volume 20, Number 3, May/June 2001, pp.70-75.

"Predicting paroxysmal atrial fibrillation/flutter: A challenge from PhysioNet and Computers in Cardiology 2001." 29 April 2001. http://physionet.org/cinc-challenge-2001.shtml (5 Aug. 2001).

"The Source for Java™ Technology." 10 August 2001. http://java.sun.com (12 Aug. 2001).

"Webopedia™." http://www.pcwebopedia.com (13 Aug. 2001).

"WhatIs.com: Part of the TechTarget.com Network of Enterprise IT Web Sites." 5 Aug. 2001. http://whatis.com (12 August 2001).

Appendix A – Formats

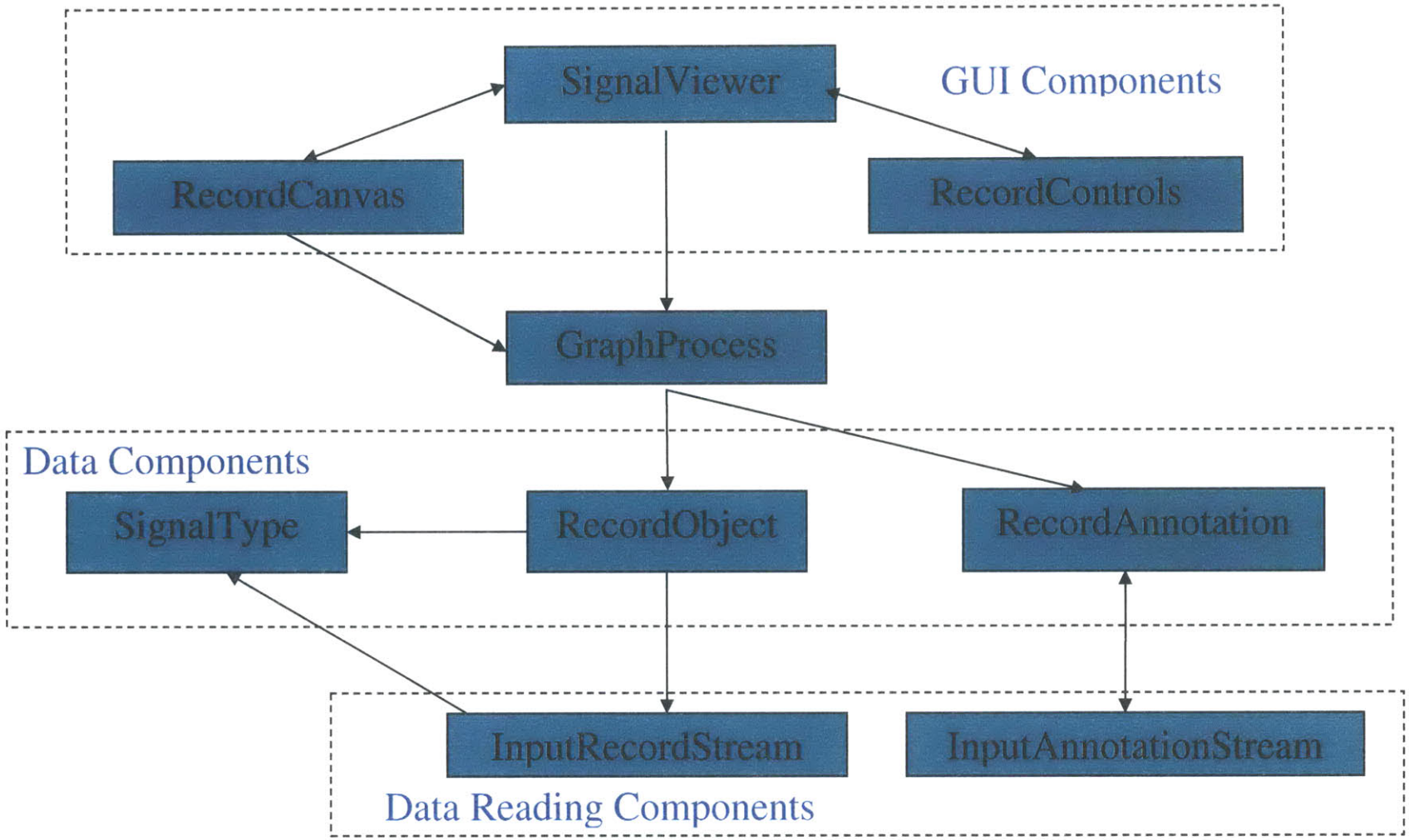
Rdsamp:
The following is the URL format for the web version of rdsamp.

```
http://www.physionet.org/cgi-bin/rdsamp
+
?database=___&record=___&start=___&end=__&doit=Show%20Samples
```

Rdann:
The following is the URL format for the web version of rdann.

```
http://www.physionet.org/cgi-bin/rdann
+
?database=___&record=___&annotator=___&tstart=___
+
&tend=___&doit=Show%20annotations
```

Appendix B – Module Dependency Diagram



Appendix C – Module Specifications

Appendix D – Java Code

Appendix C – Module Specifications

Package signalviewer

Class Summary	
AnnotationNotFoundException	This exception is thrown when an annotation type is not found in the searched portion of a record.
GraphProcess	GraphProcess is the object that is responsible for all data processing.
InputAnnotationStream	InputAnnotationStream reads annotation file data from a URL and stores it into a RecordAnnotation object.
InputRecordStream	InputRecordStream is the object that reads all the record data from a webpage and stores it into RecordObject.
RecordAnnotation	RecordAnnotation is the data object for annotation files.
RecordCanvas	This class is responsible for all graphical output on the canvas.
RecordControls	This class is responsible for all buttons and controls on the GUI.
RecordObject	This class is responsible for all graphical output on the canvas.
SignalMenu	SignalMenu extends the javax.swing.JMenu.
SignalType	SignalType encapsulates all the information of a signal.
SignalViewer	SignalViewer.java This class will extend Applet and implement the highest level container for a signal and annotation viewer.
TimeFormatException	TimeFormatException is thrown whenever the time format is incorrect.
TimeNotFoundException	TimeNotFoundException is thrown when the sample corresponding to a time is not found within a record.

signalviewer

Class AnnotationNotFoundException

`signalviewer.AnnotationNotFoundException`

---

public class **AnnotationNotFoundException**

This exception is thrown when an annotation type is not found in the searched portion of a record.

Constructor Summary

`AnnotationNotFoundException()` Creates new `AnnotationNotFoundException` without detail message.
`AnnotationNotFoundException(java.lang.String msg)` Constructs an `AnnotationNotFoundException` with the specified detail message.

Constructor Detail

AnnotationNotFoundException

`public AnnotationNotFoundException()`

Creates new `AnnotationNotFoundException` without detail message.

AnnotationNotFoundException

`public AnnotationNotFoundException(java.lang.String msg)`

Constructs an `AnnotationNotFoundException` with the specified detail message.

Parameters:

`msg` - the detail message.

---

signalviewer

Class GraphProcess

signalviewer.GraphProcess

---

public class **GraphProcess**

GraphProcess is the object that is responsible for all data processing. RecordCanvas depends on GraphProcess to provide modified data. GraphProcess is the only object to access raw data stored in RecordObject and RecordAnnotation.

Constructor Summary

GraphProcess(java.lang.String recLocURL) Constructs a GraphProcess object with only an input record.	
GraphProcess(java.lang.String recLocURL, java.lang.String recAnnURL) Constructs a GraphProcess object using both an input record and an input annotation.	

Method Summary

java.util.Vector	**getAnnotations**(int startSample, int numSamples, int screenwidth) This method returns a vector of Annotations to show within a screen.
int	**getdy**(int signal) This method returns the y difference between the actual signal data and the baseline.
float	**getGridHeight**(java.awt.Rectangle r) Returns the spacing between horizontal lines of a grid for a 1:1 sample to pixel point ratio.
float	**getGridWidth**(java.awt.Rectangle r) Returns the spacing between vertical lines, or grid width, for a 1:1 sample to pixel ratio.
int	**getNumSignals**() This method returns the number of signals in the record.
int	**getRecordLength**() This method returns the length of the record.
int	**getSample**(java.lang.String time) This method converts a string version of time in the following format: hh:mm:ss: where hh is hours, mm is minutes and ss is seconds, to the corresponding sample point.
java.lang.String	**getSignalName**(int i) This method gets the signal name of signal i.
int[]	**getSignalXComponents**(int signal, int startSample, int nsamp, int screenwidth) This method returns an array of ints that represent a single signal's x components.
int[]	**getSignalYComponents**(int signal, int startSample, int nsamp, int screenwidth, int base, float vScale) This method returns the y components of a signal.
java.lang.String	**getTime**(int sample) This method converts a sample point to the corresponding time in the string format hh:mm:ss.
int	**searchAnnotation**(int startSample, java.lang.String searchvalue, boolean direction, int nsamp, int screenwidth) This method either searches forward or backward (based on the start sample) through the annotation file for a specific annotation.

Constructor Detail

GraphProcess

```
public GraphProcess(java.lang.String recLocURL,
                    java.lang.String recAnnURL)
```

Constructs a GraphProcess object using both an input record and an input annotation.
Parameters:
`recLocURL` - Input record.
`recAnnURL` - Input annotation.

GraphProcess

```
public GraphProcess(java.lang.String recLocURL)
```

Constructs a GraphProcess object with only an input record.
Parameters:
`recLocURL` - Input record.

Method Detail

getNumSignals

```
public int getNumSignals()
```

This method returns the number of signals in the record.
Returns:
Number of signals.

---

getSignalName

```
public java.lang.String getSignalName(int i)
```

This method gets the signal name of signal i.
Parameters:
`i` - Signal ID.
Returns:
The signal name.

---

getRecordLength

```
public int getRecordLength()
```

This method returns the length of the record.
Returns:
Record length.

---

getSample

```
public int getSample(java.lang.String time)
          throws TimeFormatException
```

This method converts a string version of time in the following format: `hh:mm:ss:` where hh is hours, mm is minutes and ss is seconds, to the corresponding sample point.
Parameters:
`time` - String time input.
Returns:
The corresponding sample number.
Throws:
`TimeFormatException` - Throws exception if time format is incorrect.

---

getTime

```
public java.lang.String getTime(int sample)
```

This method converts a sample point to the corresponding time in the string format `hh:mm:ss`.
Parameters:
`sample` - Sample point.
Returns:
Time in string format.

---

getGridWidth

```
public float getGridWidth(java.awt.Rectangle r)
```

Returns the spacing between vertical lines, or grid width, for a 1:1 sample to pixel ratio.
Parameters:
`r` - The rectangle bounds of the canvas.
Returns:
The grid width.

---

getGridHeight

```
public float getGridHeight(java.awt.Rectangle r)
```

Returns the spacing between horizontal lines of a grid for a 1:1 sample to pixel point ratio.

Parameters:
`r` - Rectangle containing the bounds of the canvas.
Returns:
The fixed vertical spacing between grid lines.

---

getSignalXComponents

```
public int[] getSignalXComponents(int signal,
                                  int startSample,
                                  int nsamp,
                                  int screenwidth)
```

This method returns an array of ints that represent a single signal's x components. If there are less sample points than number of horizontal pixel points, the size of the array is the number of sample points and the output is stretched across the screen. If there are more sample points than horizontal pixel points, then the size of the array is just the number of pixel points.
Parameters:
`signal` - Signal ID.
`startSample` - The starting sample.
`nsamp` - The number of samples to display.
`screenwidth` - The width fo the screen.
Returns:
An array of ints represent the x values of a single signal.

---

getSignalYComponents

```
public int[] getSignalYComponents(int signal,
                                  int startSample,
                                  int nsamp,
                                  int screenwidth,
                                  int base,
                                  float vScale)
```

This method returns the y components of a signal. If the number of signal points to be displayed is equal to or less than the number of horizontal pixel points, then all the sample points are plotted after being scaled and adjusted to paint at the correct baseline. If the number of signal points are greater than the number of horizontal pixel points, the method uses a variation of the turning point compression to select the signal points to display. This algorithm finds the number of sample points per pixel and picks the local max or local min depending on a comparison between a previously picked sample point and the first sample point that can be potentially mapped to the next pixel.
Parameters:
`signal` - Signal ID.
`startSample` - Starting sample point.
`nsamp` - Number of samples.
`screenwidth` - Width for the screen.
`base` - Baseline of the signal.
`vScale` - Vertical scaling.
Returns:
An array of integers representing the vertical components of a signal.

---

getdy

```
public int getdy(int signal)
```

This method returns the y difference between the actual signal data and the baseline. It finds the mean of the list of sample points shown and returns it. The mean represents the difference because the screen is drawn with (0,0) at the top left corner. Thus, returning the mean returns the negative distance.
Parameters:
`signal` - Signal ID.
Returns:
The mean of the signal.

---

getAnnotations

```
public java.util.Vector getAnnotations(int startSample,
                                       int numSamples,
                                       int screenwidth)
```

This method returns a vector of Annotations to show within a screen. GetAnnotations() searches through all the annotation to find the annotations that fall between the start sample and the end sample.
Parameters:
`startSample` - Start Sample.
`numSamples` - Number of samples.
`screenwidth` - Width of the screen.
Returns:
A list of annotations.

---

searchAnnotation

```
public int searchAnnotation(int startSample,
                            java.lang.String searchvalue,
                            boolean direction,
                            int nsamp,
                            int screenwidth)
```

This method either searches forward or backward (based on the start sample) through the annotation file for a specific annotation. It returns a sample number that will center the annotation

in the canvas. Returns -1 if no annotation was found.

Parameters:

`startSample` - Start sample.
`searchvalue` - Annotation to be searched.
`direction` - The direction of the search.
`nsamp` - The number of samples.
`screenwidth` - The width of the screen.

Returns:

A new start sample.

---

signalviewer

Class InputAnnotationStream

`signalviewer.InputAnnotationStream`

---

public class **InputAnnotationStream**

InputAnnotationStream reads annotation file data from a URL and stores it into a RecordAnnotatio object.

Constructor Summary

	`InputAnnotationStream(java.lang.String input)` Constructs a new InputAnnotationStream object using a url input.

Method Summary

`void`	`storeAnn(RecordAnnotation ann)` Stores the data from the URL into the annotation object.

Constructor Detail

InputAnnotationStream

`public InputAnnotationStream(java.lang.String input)`

Constructs a new InputAnnotationStream object using a url input.

Parameters:

`input` - URL representing annotation file.

Method Detail

storeAnn

`public void storeAnn(RecordAnnotation ann)`

Stores the data from the URL into the annotation object. This method tokenizes an inputstream into lines. Then, StringTokenizer tokenizes the lines and store the data.

Parameters:

ann - RecordAnnotation storage object.

Class RecordAnnotation

`signalviewer.RecordAnnotation`

public class **RecordAnnotation**

RecordAnnotation is the data object for annotation files. This object stores all the annotations and provides methods to access them.

Constructor Summary

	RecordAnnotation() Constructs an empty annotation object for cases when no annotation files are loaded.
	RecordAnnotation(java.lang.String annString) Contructs a new RecordAnnotation.

Method Summary

void	**appendAnnType**(java.lang.String type) This method appends annotation types to the annotation vector.
void	**appendAux**(java.lang.String auxString) This method appends an auxiliary annotation to the aux vector.
void	**appendChan**(java.lang.Integer chanNum) This method appends an integer to the chan vector.
void	**appendNum**(java.lang.Integer numNum) This method appends a num to the num vector.
void	**appendSample**(java.lang.Integer samp) This method appends samples to the sample vector.
void	**appendSub**(java.lang.Integer subNum) This method appends a numeric value to the sub vector.
void	**appendTime**(java.lang.Double sec) This method appends a time to the time vector.

int	**getAnnSize()** Returns the number of annotations in file.
java.lang.String	**getAnnType(int x)** Gets an annotation type at index x.
java.lang.String	**getAux(int x)** Gets an auxiliary at index x.
java.lang.Integer	**getChan(int x)** Gets a chan type at index x.
java.lang.Integer	**getNum(int x)** Gets a num type at index x.
java.lang.Integer	**getSample(int x)** Gets sample number at index x.
java.lang.Integer	**getSub(int x)** Gets a sub type at index x.
java.lang.Double	**getTime(int x)** Gets a time at index x.
void	**printAnnType()** Prints signals stored in record object to the system.
void	**printTime()** Prints signals stored in record object to system.

Constructor Detail

RecordAnnotation

`public RecordAnnotation(java.lang.String annString)`

Contructs a new RecordAnnotation. Creates vectors for each annotation variable and then initializes the InputAnnotationStream object to get and store data.
Parameters:
`annString` - Annotation file URL.

RecordAnnotation

`public RecordAnnotation()`

Constructs an empty annotation object for cases when no annotation files are loaded.

Method Detail

appendTime

`public void appendTime(java.lang.Double sec)`

This method appends a time to the time vector.
Parameters:
`sec` - Time in seconds.

---

appendSample

`public void appendSample(java.lang.Integer samp)`

This method appends samples to the sample vector.
Parameters:
`samp` - Sample.

---

appendAnnType

`public void appendAnnType(java.lang.String type)`

This method appends annotation types to the annotation vector.
Parameters:
`type` - Annotation Types (String value).

---

appendSub

`public void appendSub(java.lang.Integer subNum)`

This method appends a numeric value to the sub vector.
Parameters:
`subNum` - Sub.

---

appendChan

`public void appendChan(java.lang.Integer chanNum)`

This method appends an integer to the chan vector.
Parameters:
`chanNum` - Chan.

---

appendNum

```
public void appendNum(java.lang.Integer numNum)
```

This method appends a num to the num vector.
Parameters:
`numNum` - The num value.

---

appendAux

```
public void appendAux(java.lang.String auxString)
```

This method appends an auxiliary annotation to the aux vector.
Parameters:
`auxString` - Auxiliary to append.

---

getTime

```
public java.lang.Double getTime(int x)
```

Gets a time at index x.
Parameters:
`x` - index.
Returns:
Time value.

---

getSample

```
public java.lang.Integer getSample(int x)
```

Gets sample number at index x.
Parameters:
`x` - index.
Returns:
Sample Number.

---

getAnnType

```
public java.lang.String getAnnType(int x)
```

Gets an annotation type at index x.
Parameters:
`x` - index.
Returns:
annotation type.

---

getSub

```
public java.lang.Integer getSub(int x)
```

Gets a sub type at index x.
Parameters:
`x` - index.
Returns:
sub type.

---

getChan

```
public java.lang.Integer getChan(int x)
```

Gets a chan type at index x.
Parameters:
`x` - index.
Returns:
chan type.

---

getNum

```
public java.lang.Integer getNum(int x)
```

Gets a num type at index x.
Parameters:
`x` - index.
Returns:
num type.

---

getAux

```
public java.lang.String getAux(int x)
```

Gets an auxiliary at index x.

Parameters:
x - index.
Returns:
aux type.

---

getAnnSize

`public int getAnnSize()`

Returns the number of annotations in file.
Returns:
number of annotations.

---

printTime

`public void printTime()`

Prints signals stored in record object to system. This method is provided as a testing tool.

---

printAnnType

`public void printAnnType()`

Prints signals stored in record object to the system. This method is provided as a testing tool.

---

signalviewer

Class RecordCanvas

`signalviewer.RecordCanvas`

---

public class **RecordCanvas**

This class is responsible for all graphical output on the canvas. It will include a basic paint for grids, signals, and annotations

Since:
August 7, 2001

Field Summary

float	**hScale** This float variable controls the horizontal scale factor.
boolean	**show_annotation** Boolean toggle variable for displaying annotations and corresponding functions.
boolean	**show_grid** Toggle variable for showing the grid.
boolean	**show_signal_name** Boolean toggle variable for showing signal names.
float	**vScale** This float variable determines the vertical scale factor.

Constructor Summary

RecordCanvas(GraphProcess gp) Corrects a new RecordCanvas by initializing with a data processing object.

Method Summary

int	**getBaseLine**(int signal) This method takes an integer representing a signal to be shown on the canvas and returns the baseline for that signal.
void	**paint**(java.awt.Graphics g) Overrides the paint method for this component.
void	**paintAnnotation**(java.awt.Graphics g, int startSamp) Paints all annotations using processed data from graphProcess.
void	**paintGrid**(java.awt.Graphics g) Paints a light gray grid that scales according to the time and amplitude scale.
void	**paintSignal**(java.awt.Graphics g, int startSamp) Paints all signals using processed data from GraphProcess.
void	**paintSignalName**(java.awt.Graphics g) Paints the signal names above the signal baselines.
void	**paintTime**(java.awt.Graphics g) This method paints the start time and the end time at the bottom corners of the canvas.
void	**redraw**(boolean sG, boolean sS, boolean sSN, boolean sA, float vMag, float hMag, int newSample) This is the basic redraw method.
void	**redraw**(java.lang.String newTime) Redraws the canvas using a string input that represents start time.
void	**redrawAnnotation**(java.lang.String searchString, boolean direction) This method provides a specific redraw method for repainting the screen based on displaying certain annotations.
void	**redrawScale**(float vMag, float hMag) This method redraws a canvas based on new horizontal and vertical scaling inputs.

Field Detail

show_grid

public boolean **show_grid**

Toggle variable for showing the grid.

---

show_annotation

public boolean **show_annotation**

Boolean toggle variable for displaying annotations and corresponding functions.

---

show_signal_name

public boolean **show_signal_name**

Boolean toggle variable for showing signal names.

---

vScale

public float **vScale**

This float variable determines the vertical scale factor.

---

hScale

public float **hScale**

This float variable controls the horizontal scale factor.

Constructor Detail

RecordCanvas

public **RecordCanvas**(GraphProcess gp)

Corrects a new RecordCanvas by initializing with a data processing object.

Parameters:

gp - The gp object will give RecordCanvas all the necessary information for painting objects to screen.

Method Detail

paint

public void **paint**(java.awt.Graphics g)

Overrides the paint method for this component. This paint method will draw grids, signals, and annotations if requested.

Parameters:

g - The graphics component to modify.

redraw

```
public void redraw(java.lang.String newTime)
            throws TimeFormatException,
                   TimeNotFoundException
```

Redraws the canvas using a string input that represents start time. This method actually calls a method to convert the string time into an int representing the start sample. Then, redraw(String newTime) calls another redraw method that actually repaints the object.

Parameters:

`newTime` - Represents the new start time on the canvas screen.

Throws:

`TimeFormatException` - Thrown if the input is incorrectly formatted.

`TimeNotFoundException` - Thrown if the time requested is not found in the record.

---

redrawScale

```
public void redrawScale(float vMag,
                        float hMag)
```

This method redraws a canvas based on new horizontal and vertical scaling inputs. After the hscale and vscale variables are set, the method calls the general redraw() method.

Parameters:

`vMag` - New vertical scale input.

`hMag` - New horizontal scale input.

---

redrawAnnotation

```
public void redrawAnnotation(java.lang.String searchString,
                             boolean direction)
                     throws AnnotationNotFoundException
```

This method provides a specific redraw method for repainting the screen based on displaying certain annotations.

Parameters:

`searchString` - Annotation search value.

`direction` - Search direction.

Throws:

`AnnotationNotFoundException` - Throws exception when the annotation search string is not found the the portion of file searched.

---

redraw

```
public void redraw(boolean sG,
                   boolean sB,
                   boolean sSN,
                   boolean sA,
                   float vMag,
                   float hMag,
                   int newSample)
            throws TimeNotFoundException
```

This is the basic redraw method. It draws the new canvas based on toggle inputs, magnification changes, and new screen starting samples.

Parameters:

`sG` - Show grid.

`sB` - Show Baselines.

`sSN` - Show Signal Names.

`sA` - Show Annotations.

`vMag` - Vertical scale.

`hMag` - Horizontal Scale.

`newSample` - New start sample.

Throws:

`TimeNotFoundException` - Throws exception when time is not found in record.

---

paintGrid

```
public void paintGrid(java.awt.Graphics g)
```

Paints a light gray grid that scales according to the time and amplitude scale. The spacing between vertical lines will always represent 0.2 seconds. The spacing between horizontal lines will always represent 0.5 mV.

Parameters:

`g` - Graphics component to modify.

---

paintSignal

```
public void paintSignal(java.awt.Graphics g,
                        int startSamp)
```

Paints all signals using processed data from GraphProcess. PaintSignal paints only a screen's length of the record starting from the sample number input.

Parameters:

`g` - Graphics component.

`startSamp` - The starting sample.

---

paintSignalName

```
public void paintSignalName(java.awt.Graphics g)
```

Paints the signal names above the signal baselines. This method calls graphProcess to get the names.
Parameters:
`g` - Graphical component.

---

paintTime

```
public void paintTime(java.awt.Graphics g)
```

This method paints the start time and the end time at the bottom corners of the canvas. PaintTime() calls a graphProcess method to get the time in the correct format.
Parameters:
`g` - Graphical component.

---

paintAnnotation

```
public void paintAnnotation(java.awt.Graphics g,
                            int startSamp)
```

Paints all annotations using processed data from graphProcess. The text annotations are always painted in the horizontal middle of the canvas and correspoding vertical lines are drawn to more clearly denote the time of the annotation. Annotations are also painted based on the starting sample.
Parameters:
`g` - Graphical component.
`startSamp` - Starting sample.

---

getBaseLine

```
public int getBaseLine(int signal)
```

This method takes an integer representing a signal to be shown on the canvas and returns the baseline for that signal.
Parameters:
`signal` - Signal number.
Returns:
Baseline for signal.

---

signalviewer

Class RecordControls

`signalviewer.RecordControls`

---

public class **RecordControls**

This class is responsible for all buttons and controls on the GUI. The RecordControl class takes input from the RecordCanvas and allows the user interaction with the display of the signals and annotations. RecordControls contains all the controls for navigation and search.

Constructor Summary

`RecordControls(RecordCanvas canvas)` Constructs a new control panel.

Method Summary

void	`setAnnotationButtons(boolean value)` This method will enable/disable the Annotation buttons based on whether annotations are displayed to screen or not.

Constructor Detail

RecordControls

```
public RecordControls(RecordCanvas canvas)
```

Constructs a new control panel. Record canvas initializes two sets of buttons: The first is for navigation by time. The second set are forward and backward search for annotations.
Parameters:
`canvas` - Canvas component to modify.

Method Detail

setAnnotationButtons

`public void setAnnotationButtons(boolean value)`

This method will enable/disable the Annotation buttons based on whether annotations are displayed to screen or not. If there are no annotations loaded, the annotation buttons may not be used.

Parameters:

`value` - User-specified annotation search value.

signalviewer

Class RecordObject

signalviewer.RecordObject

public class **RecordObject**

This class is responsible for all graphical output on the canvas. It will include a basic paint for grids, signals and annotations.

Field Summary

`java.lang.String`	**recordName** The name of the record.
`float`	**sampFreq** The sample frequency of the record.

Constructor Summary

RecordObject`(java.lang.String inputString)`
Constructs a record object and initializes the record data reading object.

Method Summary

void	**appendSignal**(SignalType signal) This method adds a new object representing a signal to the list that recordObject's list of signals.
void	**appendTime**(java.lang.Double seconds) This method appends a new time to the time vector.
java.lang.Double	**getElementAt**(int signal, int element) This method returns the requested data point of a given signal.
java.lang.String	**getName**()
int	**getNumSignals**()
float	**getSamplingFrequency**()
SignalType	**getSignalAt**(int i) This method returns the signalobject at index i.
int	**getSignalLength**(int signal)
java.lang.String	**getSignalName**(int signal)
java.lang.Double	**getTimeAt**(int i)
boolean	**isComplete**()
void	**printTime**()
boolean	**update**()

Field Detail

recordName

public java.lang.String **recordName**

The name of the record.

---

sampFreq

public float **sampFreq**

The sample frequency of the record.

Constructor Detail

RecordObject

public **RecordObject**(java.lang.String inputString)

Constructs a record object and initializes the record data reading object. InputRecordStream stores all the data within record. Record object then stores the sampling frequency. Record Object also keeps the name of the record and a list of the signals within the record.

Parameters:

inputString - url string including variables concerning the record.

Method Detail

appendSignal

public void **appendSignal**(SignalType signal)

This method adds a new object representing a signal to the list that recordObject's list of signals.

Parameters:

signal - A signal object.

---

appendTime

public void **appendTime**(java.lang.Double seconds)

This method appends a new time to the time vector.

Parameters:

seconds - Time in seconds.

---

getSignalAt

public SignalType **getSignalAt**(int i)

This method returns the signalobject at index i.

Parameters:

i - index of signal.

Returns:

An object representing a signal.

---

getElementAt

```
public java.lang.Double getElementAt(int signal,
                                     int element)
                              throws java.lang.ArrayIndexOutOfBoundsException
```

This method returns the requested data point of a given signal.

Parameters:

`signal` - Signal index.

`element` - Element of signal to access.

Returns:

Data at that point in the signal.

Throws:

`java.lang.ArrayIndexOutOfBoundsException` - Throws exception when there is no data at the specified element of the specified signal.

---

getTimeAt

```
public java.lang.Double getTimeAt(int i)
```

---

getNumSignals

```
public int getNumSignals()
```

---

getSamplingFrequency

```
public float getSamplingFrequency()
```

---

update

```
public boolean update()
```

---

isComplete

```
public boolean isComplete()
```

---

getName

```
public java.lang.String getName()
```

---

getSignalName

```
public java.lang.String getSignalName(int signal)
```

---

getSignalLength

```
public int getSignalLength(int signal)
```

---

printTime

```
public void printTime()
```

---

signalviewer

Class SignalMenu

`signalviewer.SignalMenu`

public class **SignalMenu**

SignalMenu extends the javax.swing.JMenu. This class is used as a bug fix in order to show the menu when heavy and light components are mixed in the GUI. SignalMenu sets the JMenu to use heavy components instead of light ones.

Constructor Summary

`SignalMenu()` Creates new SignalMenu

Constructor Detail

SignalMenu

`public SignalMenu()`

Creates new SignalMenu

signalviewer

Class SignalType

`signalviewer.SignalType`

public class **SignalType**

SignalType encapsulates all the information of a signal. It stores a single signal's data and provides methods to access the data. This object also operates as a virtual array.

Constructor Summary

`SignalType(java.lang.String name)` Constructs a SignalType object using a Vector.

Method Summary

void	**add**(java.lang.Double dPt) Inserts datapoint to the end of SignalType.
void	**add**(int index, java.lang.Double dPt) Inserts datapoint at given index.
void	**addElement**(java.lang.Double dPt) Appends the data point to the end of signalVector.
java.lang.Double	**getElementAt**(int i) Returns the element at signalVector[i].
java.lang.String	**getInfo**() Returns info about the class SignalType.
int	**getLength**() Returns the number of points in the signal.
java.lang.String	**getName**() Returns the name of a Signal.
void	**getSamples**(int i, int n) Returns n samples begining from index i.
int	**getSignalSize**() Returns signal size.
java.lang.Double	**getStartPoint**() Returns the beginning point of the signal.
void	**printSignal**() Prints the signal datapoints to screen.

Constructor Detail

SignalType

```
public SignalType(java.lang.String name)
```

Constructs a SignalType object using a Vector.
Parameters:
name - The name of the signal.

Method Detail

getName

```
public java.lang.String getName()
```

Returns the name of a Signal.
Returns:
signal name.

---

addElement

```
public void addElement(java.lang.Double dPt)
```

Appends the data point to the end of signalVector.
Parameters:
dPt - Data point to append.

---

add

```
public void add(int index,
                java.lang.Double dPt)
```

Inserts datapoint at given index.
Parameters:
index - Vector index.
dPt - Datapoint

---

add

```
public void add(java.lang.Double dPt)
```

Inserts datapoint to the end of SignalType.
Parameters:
dPt - Datapoint.

---

getElementAt

```
public java.lang.Double getElementAt(int i)
```

Returns the element at signalVector[i].
Parameters:
i - index
Returns:
Element at index i.

---

getSignalSize

```
public int getSignalSize()
```

Returns signal size.
Returns:
Signal size.

---

getSamples

```
public void getSamples(int i,
                       int n)
```

Returns n samples begining from index i.
Parameters:
`i` - index
`n` - number of samples

---

getStartPoint

```
public java.lang.Double getStartPoint()
```

Returns the beginning point of the signal.
Returns:
The first datapoint of the signal.

---

getLength

```
public int getLength()
```

Returns the number of points in the signal.
Returns:
signal length

---

printSignal

```
public void printSignal()
```

Prints the signal datapoints to screen. Used for testing purposes.

---

getInfo

```
public java.lang.String getInfo()
```

Returns info about the class SignalType.
Returns:
SignalType info.

---

signalviewer

Class SignalViewer

signalviewer.SignalViewer

---

public class **SignalViewer**

SignalViewer.java This class will extend Applet and implement the highest level container for a signal and annotation viewer. The class will instantiate 4 other custom classes in order to display a canvas and controls as well as read data and annotation files. The Signal Viewer class is constructed by taken two arguments of the form:
`http://www.physionet.org/cgi-bin/rdsamp?database=___&record=____&start=____&end=____`
`http://www.physionet.org/cgi-bin/rdann?database=___&record=____&annotator=____tstart`

Since:
August 10, 2001

Constructor Summary

SignalViewer() This constructor currently has no function.

Method Summary

java.lang.String	**getLabel()** This method provides a description of the applet SignalViewer
void	**init()** Init() initializes the applet.

Constructor Detail

SignalViewer

`public SignalViewer()`

This constructor currently has no function.

Method Detail

init

`public void init()`

Init() initializes the applet. This method constructs the GUI by creating a menu bar, a RecordCanvas object, and a RecordControls object. The init method also initializes a intermediary GraphProcess object in order to access record and annotation data.

---

getLabel

`public java.lang.String getLabel()`

This method provides a description of the applet SignalViewer
Returns:
A String description of the applet SignalViewer.

---

signalviewer

Class TimeFormatException

signalviewer.TimeFormatException

---

public class **TimeFormatException**

TimeFormatException is thrown whenever the time format is incorrect.

Constructor Summary

TimeFormatException() Creates new TimeFormatException without detail message.
TimeFormatException(java.lang.String msg) Constructs an TimeFormatException with the specified detail message.

Constructor Detail

TimeFormatException

public **TimeFormatException()**

Creates new TimeFormatException without detail message.

TimeFormatException

public **TimeFormatException(java.lang.String msg)**

Constructs an TimeFormatException with the specified detail message.
Parameters:
msg - the detail message.

---

signalviewer

Class TimeNotFoundException

signalviewer.TimeNotFoundException

---

public class **TimeNotFoundException**

TimeNotFoundException is thrown when the sample corresponding to a time is not found within a record.

Constructor Summary

TimeNotFoundException() Creates new TimeNotFoundException without detail message.
TimeNotFoundException(java.lang.String msg) Constructs an TimeNotFoundException with the specified detail message.

Constructor Detail

TimeNotFoundException

public **TimeNotFoundException()**

Creates new TimeNotFoundException without detail message.

TimeNotFoundException

public **TimeNotFoundException(java.lang.String msg)**

Constructs an TimeNotFoundException with the specified detail message.
Parameters:
msg - the detail message.

Appendix D – Java Code


```
/*
 * AnnotationNotFoundException.java
 *
 * Created on July 30, 2001, 12:27 PM
 */

package signalviewer;

/** This exception is thrown when an annotation type is not found in the
 * searched portion of a record.
 */
public class AnnotationNotFoundException extends java.lang.Exception {

  /**
 * Creates new <code>AnnotationNotFoundException</code> without detail message.
  */
  public AnnotationNotFoundException() {
  }


  /**
 * Constructs an <code>AnnotationNotFoundException</code> with the specified detail message.
  * @param msg the detail message.
  */
  public AnnotationNotFoundException(String msg) {
    super(msg);
  }
}
```

```
/*
 * SignalProcess.java
 *
 * Created on July 17, 2001, 1:35 PM
 */
package signalviewer;
import java.awt.*;
import java.util.*;

/** GraphProcess is the object that is responsible for all data processing.
 * RecordCanvas depends on GraphProcess to provide modified data.  GraphProcess
 * is the only object to access raw data stored in RecordObject and RecordAnnotation.
 */
public class GraphProcess extends java.lang.Object {
  // variables
  RecordObject recObj;
  RecordAnnotation ann;
  // constructors
/** Constructs a GraphProcess object using both an input record and
 * an input annotation.
 * @param recLocURL Input record.
 * @param recAnnURL Input annotation.
 */
  public GraphProcess(String recLocURL, String recAnnURL) {
    recObj = new RecordObject(recLocURL);
    ann = new RecordAnnotation(recAnnURL);
  }

/** Constructs a GraphProcess object with only an input record.
 * @param recLocURL Input record.
 */
  public GraphProcess(String recLocURL) {
    recObj = new RecordObject(recLocURL);
    ann = new RecordAnnotation();
  }
  // methods
/** This method returns the number of signals in the record.
 * @return Number of signals.
 */
  public int getNumSignals() {
    int temp = recObj.getNumSignals();
    return temp;
  }

/** This method gets the signal name of signal i.
 * @param i Signal ID.
 * @return The signal name.
 */
  public String getSignalName(int i) {
    String name = recObj.getSignalName(i);
    return name;
  }

/** This method returns the length of the record.
 * @return Record length.
```



```
 */
  public int getRecordLength() {
    return recObj.getSignalLength(0);
  }

/** This method converts a string version of time in the following format:
 * <CODE>hh:mm:ss:</CODE>
 * where hh is hours, mm is minutes and ss is seconds, to the corresponding
 * sample point.
 *
 * @param time String time input.
 * @throws TimeFormatException Throws exception if time format is incorrect.
 * @return The corresponding sample number.
 */
  public int getSample(String time) throws TimeFormatException {
    int hour, minute, second, timeinMS, samp;
    String sHour, sMin, sSec;
    StringTokenizer sT = new StringTokenizer(time, ":");
    //Integer t = new Integer(time);
    try {
      sHour = sT.nextToken();
      sMin = sT.nextToken();
      sSec = sT.nextToken();
      //System.out.println(sHour + , + sMin + , +sSec);
    } catch(NoSuchElementException e) {throw new TimeFormatException();}
    try {
      hour = (new Integer(sHour)).intValue();
      minute = (new Integer(sMin)).intValue();
      second = (new Integer(sSec)).intValue();
      //System.out.println(hour + , + minute + , + second);
    } catch (NumberFormatException e) {throw new TimeFormatException();}
    timeinMS = (3600*hour + 60*minute + second) * 1000;
    samp = (int) (timeinMS / recObj.sampFreq);

    //System.out.println(hour +, + minute + , + second);
    return samp;
  }

/** This method converts a sample point to the corresponding time in the string
 * format <CODE>hh:mm:ss</CODE>.
 * @param sample Sample point.
 * @return Time in string format.
 */
  public String getTime(int sample) {
    int time, millisecond, second, minute, hour;
    float sf = recObj.sampFreq;
    String sHour, sMin, sSec, sMSec, sTime;
    //float second, minute, hour, millisecond;

    time = (int) (sf*sample);
    second = time/1000;
    millisecond = time % 1000;
    minute = second/60;
    if (minute > 0) {
      second = second - (minute*60);
```

```
        hour = minute/60;
        if (hour >0) {
            minute = minute - (hour*60);
        }
    } else hour = 0;
    sHour = "" + hour;
    sMin = "" + minute;
    sSec = "" + second;
    sMSec = "" + millisecond;
    if (hour < 10)
      sHour = "0" + sHour;
    if (minute < 10)
        sMin = "0" + sMin;
    if (second < 10)
        sSec = "0" + sSec;
    //System.out.println(Time: + time);
    sTime = sHour + ":" + sMin + ":" + sSec + "." +sMSec;
    return sTime;
  }

  // methods pertaining to grid
/** Returns the spacing between vertical lines, or grid width, for a 1:1
 * sample to pixel ratio.
 * @param r The rectangle bounds of the canvas.
 * @return The grid width.
 */
  public float getGridWidth(Rectangle r) {
      float gWidth, seconds;
      int cwidth = r.width;
      float sfreq = recObj.sampFreq/1000f;

      seconds = r.width * sfreq;    // seconds = hMax
      gWidth = cwidth/seconds;      // pixel per second
      gWidth /= 5; // get correct (0.2 seconds) grid interval
      return gWidth;
  }

/** Returns the spacing between horizontal lines of a grid for a 1:1 sample to
 * pixel point ratio.
 *
 * @param r Rectangle containing the bounds of the canvas.
 * @return The fixed vertical spacing between grid lines.
 */
  public float getGridHeight(Rectangle r) {
      float gHeight, millivolts;
      int cheight = r.height;

      millivolts = (r.height/1000f);   // millivolts = vMax
      gHeight = cheight/millivolts;    // pixel/millivolt
      //System.out.println(gHeight: + gHeight);
      gHeight /= 2;
      return gHeight;
  }

  // methods pertaining to signals
```



```
/** This method returns an array of ints that represent a single signal's
 * x components. If there are less sample points than number of horizontal
 * pixel points, the size of the array is the number of sample points and the
 * output is stretched across the screen. If there are more sample points than
 * horizontal pixel points, then the size of the array is just the number
 * of pixel points.
 * @param signal Signal ID.
 * @param startSample The starting sample.
 * @param nsamp The number of samples to display.
 * @param screenwidth The width fo the screen.
 * @return An array of ints represent the x values
 * of a single signal.
 */
  public int[] getSignalXComponents (int signal, int startSample, int nsamp, int screenwidth){
      int arraysize, x;
      float hscale = 1f;

      if (nsamp ≤ screenwidth) {
          arraysize = nsamp;
          hscale = screenwidth/((float)arraysize); }
      else arraysize = screenwidth;

      int[] temp = new int[arraysize];
      for(int i = 0; i < arraysize; i++) {
          temp[i] = (int)(i*hscale + 0.5);
      }
      return temp;
  }

/** This method returns the y components of a signal. If the number of signal
 * points to be displayed is equal to or less than the number of horizontal
 * pixel points, then all the sample points are plotted after being scaled
 * and adjusted to paint at the correct baseline. If the number of signal
 * points are greater than the number of horizontal pixel points, the method
 * uses a variation of the turning point compression to select the signal points
 * to display. This algorithm finds the number of sample points per pixel and
 * picks the local max or local min depending on a comparison between a previously
 * picked sample point and the first sample point that can be potentially mapped
 * to the next pixel.
 * @param signal Signal ID.
 * @param startSample Starting sample point.
 * @param nsamp Number of samples.
 * @param screenwidth Width for the screen.
 * @param base Baseline of the signal.
 * @param vScale Vertical scaling.
 * @return An array of integers representing the vertical
 * components of a signal.
 */
  public int[] getSignalYComponents (int signal, int startSample, int nsamp, int screenwidth, int base, float vScale) {
      int arraysize;
      int x, y, z, dy;  // dy is displacement of signal as compared to baseline
      int[] temp;
      float fsampPerPixel;

      if (nsamp ≤ screenwidth) {
```

```
        arraysize = nsamp;
        temp = new int[arraysize];
        dy = getdy(signal);
        for(int i = 0; i < arraysize; i++) {
            x = -recObj.getElementAt(signal, startSample + i).intValue();
            temp[i] = base + ((int)((dy + x) * vScale));
        }
    }
    else {
        arraysize = screenwidth;
        temp = new int[arraysize];

        int tempSamp = startSample;
        x = recObj.getElementAt(signal, tempSamp).intValue();
        temp[0] = -x;
        fsampPerPixel = (nsamp/(float)screenwidth);

        float old_tot, new_tot;
        old_tot = 0f;
        for (int i = 0; i < arraysize; i++) {
            int sampPerPixel;
            new_tot = (i+1) * fsampPerPixel;
            sampPerPixel = (int)(new_tot+0.5) - (int)(old_tot+0.5);
            old_tot = new_tot;
            //try {
                y = recObj.getElementAt(signal, tempSamp + 1). intValue();
            //}catch (java.lang.ArrayIndexOutOfBoundsException e) {}
            if (y ≥ x) {
                // if y ≥ x, look for local max
                for (int n=2; n < sampPerPixel; n++) {
                    z = recObj.getElementAt(signal, tempSamp+n).intValue();
                    if (z ≥ y)
                        y = z;}
            }
            else {
                // if y < x, look for local min
                for (int n=2; n < sampPerPixel; n++) {
                    z = recObj.getElementAt(signal, tempSamp+n).intValue();
                    if (z ≤ y)
                        y = z;}
            }
            x = y;

            temp[i] = -x;
            tempSamp = tempSamp + sampPerPixel;
        } // end for loop


        dy = getdy(signal);
        for (int i = 0; i < arraysize; i++) {
            if (temp[i] > 1000)
                temp[i] /= 1000;
            temp[i] = base + ((int)((dy + temp[i])*vScale));
        }
    } // end else
```



```
        return temp;
    }

    /** This method returns the y difference between the actual signal
     * data and the baseline. It finds the mean of the list of sample
     * points shown and returns it. The mean represents the difference
     * because the screen is drawn with (0,0) at the top left corner.
     * Thus, returning the mean returns the negative distance.
     * @param signal Signal ID.
     * @return The mean of the signal.
     */
    public int getdy(int signal) {

        int mean;
        int sum = 0;
        int nsamp = 1000;

        for (int i = 0; i < nsamp; i++) {
            sum += recObj.getElementAt(signal, i).intValue(); }
        mean = sum/nsamp;
        return mean;
    }

    // methods pertaining to annotations
    /** This method returns a vector of Annotations to show within a screen.
     * GetAnnotations() searches through all the annotation to find the
     * annotations that fall between the start sample and the end sample.
     * @param startSample Start Sample.
     * @param numSamples Number of samples.
     * @param screenwidth Width of the screen.
     * @return A list of annotations.
     */
    public Vector getAnnotations(int startSample, int numSamples, int screenwidth) {
        int endSample = startSample + numSamples;
        int nAnn = ann.getAnnSize();
        Vector position = new Vector();
        Vector sample = new Vector();
        Vector annType = new Vector();
        Vector sub, chan, num, aux = new Vector();
        Integer samp;
        int s;
        String aType = "";
        float hScale = 1f;

        hScale = screenwidth/((float)numSamples);
        for (int i = 0; i < nAnn; i++) {
            samp = ann.getSample(i);
            if ((samp.intValue() ≥ startSample) && (samp.intValue() ≤ endSample)) {
                s = (int) ((float)(samp.intValue() - startSample) * hScale + 0.5);
                sample.add(new Integer(s));
                aType = ann.getAnnType(i);
                if (aType.equals("+")) {
                    aType = ann.getAux(i);}
                annType.add(aType);
            }
```

```
        if (samp.intValue() > endSample) break;
      }
      position.add(sample);
      position.add(annType);
      return position;
    }

/** This method either searches forward or backward (based on the start sample)
 * through the annotation file for a specific annotation. It returns a sample
 * number that will center the annotation in the canvas. Returns -1 if no
 * annotation was found.
 * @param startSample Start sample.
 * @param searchvalue Annotation to be searched.
 * @param direction The direction of the search.
 * @param nsamp The number of samples.
 * @param screenwidth The width of the screen.
 * @return A new start sample.
 */
    public int searchAnnotation(int startSample, String searchvalue, boolean direction, int nsamp, int screenwidth) {
      int temp, position, offset;
      float hScale;
      int sample = -1;
      int nAnn = ann.getAnnSize();
      int recLength = getRecordLength();

      hScale = nsamp/(float)screenwidth;
      //System.out.println(hScale: + hScale);
      offset = (int)((screenwidth/2f)*hScale);
      //System.out.println(startSample: + startSample);
      if (direction) {
        for (int i = 0; i < nAnn; i++) {
          temp = ann.getSample(i).intValue();
          if ((temp > (startSample + offset))
            && (temp < recLength)
            && (searchvalue.equals(ann.getAnnType(i)))) {
            sample = temp;
            //System.out.println(sample in for loop: + sample);
            break;
          }
        }
      } else {
        for (int j = (nAnn - 1); j ≥ 0; j--) {
          temp = ann.getSample(j).intValue();
          if ((temp < (startSample + offset))
            && (temp < recLength)
            && (searchvalue.equals(ann.getAnnType(j)))) {
            sample = temp;
            break;
          }
        }
      }
      if (sample < 0)
        return sample;
      else {
        position = sample - offset;
```



```
        if (position < 0)
          position = 0;
        int t = position - sample;
        return position;
      }
    } // end searchAnnotation
}
```

```
/*
 * InputAnnotationStream.java
 *
 * Created o May 3, 2001, 1:20 PM
 * purpose:
 */

package signalviewer;
import java.util.*;
import java.net.*;
import java.io.*;


/** InputAnnotationStream reads annotation file data from a URL and stores it
 * into a RecordAnnotation object.
 */
public class InputAnnotationStream extends Thread {
  // variables
    private URL annURL;
    private InputStream inStream;
    private StreamTokenizer in;
    private BufferedReader r;

  // constructors
/** Constructs a new InputAnnotationStream object using a url input.
 * @param input URL representing annotation file.
 */
    public InputAnnotationStream(String input) {
        r = convertStringtoReader(input);
    }

  // methods
    private BufferedReader convertStringtoReader(String input) {
    // input: String URL
    // output: the same URL data as a Stream Tokenizer
    // description: This method converts a url in string form to java URL form
    //              and finally to a StreamTokenizer.  This stream of data is
    //              now fetchable as tokens.
        System.out.println("*********ann convertReader reached **");
        try {annURL = new java.net.URL(input);}
        catch (MalformedURLException e) {System.out.println("MalFormedURLException");}
        try {inStream = annURL.openStream();}
        catch (IOException e) {System.out.println("URL Stream error");}

        //try {FileInputStream inStream = new FileInputStream(input);}
        //catch (FileNotFoundException e) {System.out.println(file not found);}
        //BufferedInputStream bIn = new BufferedInputStream(inStream);
        BufferedReader r = new BufferedReader(new InputStreamReader(inStream));
        return r;
    }

/** Stores the data from the URL into the annotation object.  This method tokenizes
 * an inputstream into lines.  Then, StringTokenizer tokenizes the lines and store
 * the data.
 * @param ann RecordAnnotation storage object.
```



```
 */
    public void storeAnn(RecordAnnotation ann) {
        System.out.println("storeAnn reached");
        double minInMil;  // minutes in millisecs
        double secInMil;          // seconds in millisecs
        double millisec;
        Double time; // use millisecs
        String sTime;
        Integer sample;
        String anntype;
        Integer sub;
        Integer chan;
        Integer num;
        String aux;
        boolean flag = true;
        int n = 0;
        String line = null;

        try {
            line = r.readLine();}
        catch (IOException e) {System.out.println("rdAnn readline failed");}
        int cnt = 0;
        while (line != null) {
            cnt++;
            //System.out.println(line);
            StringTokenizer st = new StringTokenizer(line);
            sTime = st.nextToken();
            StringTokenizer stTime = new StringTokenizer(sTime, "[]:.");
            minInMil = new Integer(stTime.nextToken()).doubleValue();
            secInMil = new Integer(stTime.nextToken()).doubleValue();
            millisec = new Integer(stTime.nextToken()).doubleValue();
            time = new Double(minInMil *60 *1000 + secInMil *1000 + millisec);
            sample =  new Integer(st.nextToken());
            anntype = st.nextToken();
            sub = new Integer(0);
            try {
                sub = new Integer(st.nextToken());
            }
            catch (NumberFormatException e) {System.out.println(cnt);}
            chan = new Integer(st.nextToken());
            num = new Integer(st.nextToken());
            if (st.hasMoreElements())
                aux = st.nextToken();
            else aux = null;

            /*System.out.println(time.toString());
            System.out.println(sample.toString());
            System.out.println(anntype);
            System.out.println(sub.toString());
            System.out.println(chan.toString());
            System.out.println(num.toString());
            System.out.println(aux);*/
            ann.appendTime(time);
            ann.appendSample(sample);
            ann.appendAnnType(anntype);
```

```
        ann.appendSub(sub);
        ann.appendChan(chan);
        ann.appendNum(num);
        ann.appendAux(aux);
        try {
          line = r.readLine();}
        catch (IOException e) {System.out.println("IOException occured");}

/*
      try {
        line = r.readLine();}
      catch (IOException e) {System.out.println(rdAnn readline failed);}
      int cnt = 0;
      while (line != null) {
        cnt++;
        //System.out.println(line);
        StringTokenizer st = new StringTokenizer(line, \t\n\r\f.);
        minInMil = new Integer(st.nextToken()).doubleValue();
        secInMil = new Integer(st.nextToken()).doubleValue();
        millisec = new Integer(st.nextToken()).doubleValue();
        time = new Double(minInMil *60 *1000 + secInMil *1000 + milliser);
        sample =  new Integer(st.nextToken());
        anntype = st.nextToken();
        sub = new Integer(0);
        try {
          sub = new Integer(st.nextToken());
        }
        catch (NumberFormatException e) {System.out.println(cnt);}
        chan = new Integer(st.nextToken());
        num = new Integer(st.nextToken());
        if (st.hasMoreElements())
          aux = st.nextToken();
        else aux = null;

        /*System.out.println(time.toString());
        System.out.println(sample.toString());
        System.out.println(anntype);
        System.out.println(sub.toString());
        System.out.println(chan.toString());
        System.out.println(num.toString());
        System.out.println(aux);*/
/*        ann.appendTime(time);
        ann.appendSample(sample);
        ann.appendAnnType(anntype);
        ann.appendSub(sub);
        ann.appendChan(chan);
        ann.appendNum(num);
        ann.appendAux(aux);
        try {
          line = r.readLine();}
        catch (IOException e) {System.out.println(IOException occured);}*/
      }
    } // storeAnn
```



```
/* private StreamTokenizer convertStringtoTokenizer(String input) {
  // input: String URL
  // output: the same URL data as a Stream Tokenizer
  // description: This method converts a url in string form to java URL form
  //              and finally to a StreamTokenizer.  This stream of data is
  //              now fetchable as tokens.
    System.out.println(*********ann convertStringtoTokenizer reached **);
    try {annURL = new java.net.URL(input);}
    catch (MalformedURLException e) {System.out.println(MalFormedURLException);}
    try {inStream = annURL.openStream();}
    catch (IOException e) {System.out.println(URL Stream error);}

    //try {FileInputStream inStream = new FileInputStream(input);}
    //catch (FileNotFoundException e) {System.out.println(file not found);}
    //BufferedInputStream bIn = new BufferedInputStream(inStream);
    Reader r = new BufferedReader(new InputStreamReader(inStream));
    StreamTokenizer sIn = new StreamTokenizer(r);
    return sIn;
  }

  public void storeAnn(RecordAnnotation ann) {
  // input:
  // output:
  // description:
    double minInMil;  // minutes in millisecs
    double secInMil;        // seconds in millisecs
    //double millisec;
    Double time; // use millisecs
    Integer sample;
    String anntype;
    Integer sub;
    Integer chan;
    Integer num;
    String aux;
    boolean flag = true;
    int n = 0;

    //in.whitespaceChars(46,46);  // sets . as delimiter
    in.whitespaceChars(58,58);  // sets : as delimiter
    in.ordinaryChar(43);        /  / sets + as an ordinary char
    try {
      in.eolIsSignificant(false);
      while (flag) {
        System.out.println(Iteration: + n);
        in.nextToken();
        minInMil = (in.nval * 60 * 1000);
        //System.out.println(minInMil: + minInMil);
        in.nextToken();
        //System.out.println(in.toString());
        secInMil = (in.nval * 1000);
        //System.out.println(secInMil: + secInMil);
        time = new Double(minInMil + secInMil);
        //System.out.println(time: + time);
        in.nextToken();
        //System.out.println(in.toString());
```

```
        sample = new Integer((int)in.nval);
        in.nextToken();
        //System.out.println(anntype: +in.toString());
        anntype = in.sval;
        if (in.ttype == 43)
           anntype = + (char) in.ttype;
        //System.out.println(anntype = +anntype);
        in.nextToken();
        //System.out.println(in.toString());
        sub = new Integer((int)in.nval);
        in.nextToken();
        //System.out.println(in.toString());
        chan = new Integer((int)in.nval);
        in.nextToken();
        //System.out.println(in.toString());
        num = new Integer((int)in.nval);
        if (anntype.equals(+)) {
           in.nextToken();
           //System.out.println(in.toString());
           in.nextToken();
           //System.out.println(in.toString());
           aux = in.sval;}
        else aux = new String();

        System.out.println(time.toString());
        System.out.println(sample.toString());
        System.out.println(anntype);
        System.out.println(sub.toString());
        System.out.println(chan.toString());
        System.out.println(num.toString());
        System.out.println(aux);

        ann.appendTime(time);
        ann.appendSample(sample);
        ann.appendAnnType(anntype);
        ann.appendSub(sub);
        ann.appendChan(chan);
        ann.appendNum(num);
        ann.appendAux(aux);
        n++;
        if (n == 50) {
           flag = false;
           System.out.println(Done with 50 annotations);}
     } // while loop
   } catch(IOException e) {System.out.println(annStore failed);}
 } // storeAnn*/

}
```

```
/*
 * InputRecordStream.java
 *
 * Created on May 8, 2001,
 * purpose:
 */

package signalviewer;
import java.util.*;
import java.io.*;
import java.net.*;

/** InputRecordStream is the object that reads all the record data from a webpage
 * and stores it into RecordObject.
 */
public class InputRecordStream extends Thread {
  // variables
    private URL inputURL;
    private InputStream inStream;
    private StreamTokenizer in;
    private BufferedReader r;

  // constructors
/** Constructs a new InputRecordStream object using a given text URL.
 * @param input Text Url.
 */
    public InputRecordStream(String input) {
      r = convertStringtoReader(input);
    }

  // methods
    private BufferedReader convertStringtoReader(String input) {
    // input: String URL
    // output: the same URL data as a Stream Tokenizer
    // description: This method converts a url in string form to java URL form
    //              and finally to a StreamTokenizer. This stream of data is
    //              now fetchable as tokens.
      System.out.println("*********convertStringtoTokenizer reached **");
      try {inputURL = new java.net.URL(input);}
      catch (MalformedURLException e) {System.out.println("MalFormedURLException");}
      try {inStream = inputURL.openStream();}
      catch (IOException e) {System.out.println("URL Stream error");}
      BufferedReader r = new BufferedReader(new InputStreamReader(inStream));
      return r;
    }

/** Initializes data reading for the record. This method converts the String
 * input into an input stream.
 * @param record The record object for data storage.
 */
    public void initSignals(RecordObject record) {
      String line = null;
      String temp = null;

      System.out.println("Record initSignals reached");
```



```
      try {
        line = r.readLine();}
      catch (IOException e) {}
      StringTokenizer st = new StringTokenizer(line);
      temp = st.nextToken();
      while (st.hasMoreElements()) {
        SignalType signal = new SignalType(st.nextToken());
        record.appendSignal(signal);
        System.out.println(signal.signalName + " appended");}
      try {
        line = r.readLine();}
      catch (IOException e) {}
    } // initSignals

/** Stores the data by tokenizing the input stream by lines. Then, StringTokenizer
 * is used to tokenize the line and the data is stored into the RecordObject.
 * @param record The record object for data storage.
 */
    public void storeData(RecordObject record) {
      System.out.println("InputRecordStream storeData reached");
      int numSigs = record.getNumSignals();
      int i;
      String line = null;
      double time1, time2, sig1, sig2;
      Double time, sig;

      try{
        line = r.readLine();}
      catch (IOException e) {System.out.println("rdSamp readline failed");}
      while (line != null) {
        //System.out.println(line);
        StringTokenizer st = new StringTokenizer(line, " \t\n\r\f(");
//        time1 = new Integer(st.nextToken()).doubleValue() * 1000;
//        time2 = new Integer(st.nextToken()).doubleValue();
//        time = new Double(time1 + time2);
        time = new Double(new Double(st.nextToken()).doubleValue() * 1000);
        //System.out.println(time);
        record.appendTime(time);
        i = 0;
        while (st.hasMoreElements()) {
          SignalType signal = record.getSignalAt(i);
//          sig1 = new Integer(st.nextToken()).doubleValue()*1000;
//          sig2 = new Integer(st.nextToken()).doubleValue();
//          sig = new Double(sig1+sig2);
          sig = new Double(new Double(st.nextToken()).doubleValue() * 1000);
        //  System.out.println(sig);
          signal.add(sig);
          i++;
        }
        try {
          line = r.readLine();}
        catch (IOException e) {System.out.println("IOException occured");}
      }
    } // storeData
```

```
/*private void storeData(StreamTokenizer in) {
// input: StreamTokenizer input
// output: none
// description:  This method takes the stream tokenizer and parses
//               through the data and stores each signal into objects
//               called SignalTypes.
   String line;
   in.eolIsSignificant(true);

      int characterType = in.ttype;
   try{
      in.nextToken();
      while (in.ttype != in.TT_EOF) {
         in.pushBack();
      //while (in.lineno() != 3602) {
         characterType = readStreamLine(in);
      //System.out.println("****line no: " + in.lineno());
         in.nextToken();}
      System.out.println(in.ttype);
      System.out.println(signalA.lastElement().toString());
      System.out.println(signalB.lastElement().toString());
      }
   catch (IOException e) {}
}

private int readStreamLine(StreamTokenizer in) throws IOException{
// input:
// ouput:
// description:

   //System.out.println("******readStreamLine reached****");
   //System.out.println("******previous token is: " + in.sval);
   try {in.nextToken();   //gets samp #
      in.nextToken();
      signalA.add(new Double(in.nval));
      in.nextToken();
      signalB.add(new Double(in.nval));
      in.nextToken();}
   catch (IOException e) {}
   if (in.ttype == in.TT_EOL) {
      //System.out.println("**** readStreamLine returns: " + in.ttype);
      return in.ttype; }
   else {
      System.out.println("file has more than 2 signals");
      throw new IOException(); }
}*/
}
```

```
/*
 * RecordAnnotation.java
 *
 * Created on May 8, 2001, 1:37 PM
 * purpose:
 */

package signalviewer;
import java.awt.Graphics;
import java.util.*;

/** RecordAnnotation is the data object for annotation files.  This object stores
 * all the annotations and provides methods to access them.
 */
public class RecordAnnotation {
  // variables
    Vector time;
    Vector sample;
    Vector anntype;
    Vector sub;
    Vector chan;
    Vector num;
    Vector aux;
    InputAnnotationStream inAnnStrm;

  // constructors
/** Contructs a new RecordAnnotation.  Creates vectors for each annotation variable
 * and then initializes the InputAnnotationStream object to get and store data.
 * @param annString Annotation file URL.
 */
    public RecordAnnotation(String annString) {
        System.out.println("Constructing Annotation:");
        time = new Vector();
        sample = new Vector();
        anntype = new Vector();
        sub = new Vector();
        chan = new Vector();
        num = new Vector();
        aux = new Vector();
        inAnnStrm = new InputAnnotationStream(annString);
        inAnnStrm.storeAnn(this);
        //printTime();
        //printAnnType();
    }

/** Constructs an empty annotation object for cases when no annotation
 * files are loaded.
 */
    public RecordAnnotation() {
    }
  // methods
/** This method appends a time to the time vector.
 * @param sec Time in seconds.
 */
    public void appendTime(Double sec){
```



```
        time.add(sec);
    }

/** This method appends samples to the sample vector.
 * @param samp Sample.
 */
    public void appendSample(Integer samp){
        sample.add(samp);
    }

/** This method appends annotation types to the annotation vector.
 * @param type Annotation Types (String value).
 */
    public void appendAnnType(String type){
        anntype.add(type);
    }

/** This method appends a numeric value to the sub vector.
 * @param subNum Sub.
 */
    public void appendSub(Integer subNum) {
        sub.add(subNum);
    }

/** This method appends an integer to the chan vector.
 * @param chanNum Chan.
 */
    public void appendChan(Integer chanNum) {
        chan.add(chanNum);
    }

/** This method appends a num to the num vector.
 * @param numNum The num value.
 */
    public void appendNum(Integer numNum) {
        num.add(numNum);
    }

/** This method appends an auxiliary annotation to the aux vector.
 * @param auxString Auxiliary to append.
 */
    public void appendAux(String auxString) {
        aux.add(auxString);
    }

/** Gets a time at index x.
 * @param x index.
 * @return Time value.
 */
    public Double getTime(int x) {
        Double t;
        t = (Double) time.elementAt(x);
        return t;
    }
```

```
/** Gets sample number at index x.
 * @param x index.
 * @return Sample Number.
 */
  public Integer getSample(int x) {
     Integer i;
     i = (Integer) sample.elementAt(x);
     return i;
  }
/** Gets an annotation type at index x.
 * @param x index.
 *@return annotation type.
 */
  public String getAnnType(int x) {
     String s;
     s = (String) anntype.elementAt(x);
     return s;
  }

/** Gets a sub type at index x.
 * @param x index.
 * @return sub type.
 */
  public Integer getSub(int x) {
     Integer i;
     i = (Integer) sub.elementAt(x);
     return i;
  }

/** Gets a chan type at index x.
 * @param x index.
 * @return chan type.
 */
  public Integer getChan(int x) {
     Integer i;
     i = (Integer) chan.elementAt(x);
     return i;
  }

/** Gets a num type at index x.
 * @param x index.
 * @return num type.
 */
  public Integer getNum(int x) {
     Integer i;
     i = (Integer) num.elementAt(x);
     return i;
  }

/** Gets an auxiliary at index x.
 * @param x index.
 * @return aux type.
 */
  public String getAux(int x) {
     String s;
```



```
     s = (String) aux.elementAt(x);
     return s;
  }

/** Returns the number of annotations in file.
 * @return number of annotations.
 */
  public int getAnnSize() {
     return sample.size();
  }

/** Prints signals stored in record object to system. This method is
 * provided as a testing tool.
 */
  public void printTime() {
     System.out.println("********** printing Time ***");
     for (Enumeration e = time.elements(); e.hasMoreElements();)
     {
        System.out.println(e.nextElement());
     }
  }// printTime

/** Prints signals stored in record object to the system. This method is
 * provided as a testing tool.
 */
  public void printAnnType() {
     System.out.println("********** printing Annotation ***");
     for (Enumeration e = anntype.elements(); e.hasMoreElements();)
     {
        System.out.println(e.nextElement());
     }
  }// printAnn
}
```

```
/*
 * RecordCanvas.java
 * Created on May 8, 2001, 1:16 PM
 * Created by Emily Liu
 */

package signalviewer;
import java.awt.*;
import java.util.*;
/** This class is responsible for all graphical output on the canvas.
 * It will include a basic paint for grids, signals, and annotations
 * @author Emily Liu
 * @since August 7, 2001
 */
public class RecordCanvas extends Canvas {

  // Variables
  GraphProcess gprocess;
/** Toggle variable for showing the grid.
 */
  public boolean show_grid = true;
/** Boolean toggle variable for displaying annotations and corresponding
 * functions.
 */
  public boolean show_annotation = true;
/** Boolean toggle variable for showing signal names.
 */
  public boolean show_signal_name = true;
  boolean show_baselines = false;
  int nSamples; // set in paint method
  int nSigs;
  int startSamp;
/** This float variable determines the vertical scale factor.
 */
  public float vScale; // = 0.05f;
/** This float variable controls the horizontal scale factor.
 */
  public float hScale;

  // Constructor(s)
/** Corrects a new RecordCanvas by initializing with a data processing object.
 * @param gp The gp object will give RecordCanvas all the necessary
 * information for painting objects to screen.
 */
  public RecordCanvas(GraphProcess gp) {
    System.out.println("Record Canvas with rec and ann");
    gprocess = gp;
    startSamp = 0;
    nSigs = gprocess.getNumSignals();
    nSamples = 0;
    vScale = 0.05f;
    hScale = 1.8f;
  }

  // Methods
```



```
/** Overrides the paint method for this component. This paint method will
 * draw grids, signals, and annotations if requested.
 * @param g The graphics component to modify.
 */
  public void paint(Graphics g) {
    Rectangle r = getBounds();
    nSamples = (int)(hScale*r.width);

    // Checks if the last sample point displayed is in the record and if
    // not, moves the screen to display the last portion of the record.
    if (this.startSamp > (gprocess.getRecordLength() - nSamples))
      this.startSamp = gprocess.getRecordLength() - nSamples;
    // Checks if the startSamp is a positive number, if not, sets to
    // beginning of the record.
    if (this.startSamp < 0)
      this.startSamp = 0;
    setBackground(java.awt.Color.white);
    System.out.println("paint method with graphics processing called");
    if (show_grid) paintGrid(g);
    if (show_baselines) paintBaseLines(g);
    if (show_signal_name) paintSignalName(g);
    paintTime(g);
    paintSignal(g, startSamp);
    if (show_annotation) paintAnnotation(g, startSamp);
  }

/** Redraws the canvas using a string input that represents start time. This
 * method actually calls a method to convert the string time into an int
 * representing the start sample. Then, redraw(String newTime) calls another
 * redraw method that actually repaints the object.
 * @param newTime Represents the new start time on the canvas screen.
 * @throws TimeFormatException Thrown if the input is incorrectly formatted.
 * @throws TimeNotFoundException Thrown if the time requested is not found in the record.
 */
  public void redraw(String newTime) throws TimeFormatException, TimeNotFoundException{
    int newSamp;
    try {
      newSamp = gprocess.getSample(newTime);}
      catch(TimeFormatException e) {throw e;}
      try {
        this.redraw(this.show_grid, this.show_baselines, this.show_signal_name, this.show_annotation, this.vScale, this.hScale
        catch(TimeNotFoundException e) {throw e;}
  }

/** This method redraws a canvas based on new horizontal and vertical scaling
 * inputs. After the hscale and vscale variables are set, the method calls
 * the general redraw() method.
 * @param vMag New vertical scale input.
 * @param hMag New horizontal scale input.
 */
  public void redrawScale(float vMag, float hMag) {
    try {
      this.redraw(this.show_grid, this.show_baselines, this.show_signal_name, this.show_annotation, vMag, hMag, this.startSa
      catch(TimeNotFoundException e) {}
```

```
  }

/** This method provides a specific redraw method for repainting the screen based
 * on displaying certain annotations.
 * @param searchString Annotation search value.
 * @param direction Search direction.
 * @throws AnnotationNotFoundException Throws exception when the annotation search string
 * is not found the the portion of file searched.
 */
  public void redrawAnnotation(String searchString, boolean direction) throws AnnotationNotFoundException{
      // input:
      // output:
      // description:
      Rectangle r = getBounds();
      int newSamp;
      newSamp = gprocess.searchAnnotation(startSamp, searchString, direction, nSamples, r.width);
      if (newSamp == -1)
         throw new AnnotationNotFoundException();
      else {
         try {this.redraw(this.show_grid, this.show_baselines, this.show_signal_name, this.show_annotation, this.vScale, this.hSc
         catch (TimeNotFoundException e) {}
      }
  }

/** This is the basic redraw method. It draws the new canvas based on toggle
 * inputs, magnification changes, and new screen starting samples.
 * @param sG Show grid.
 * @param sB Show Baselines.
 * @param sSN Show Signal Names.
 * @param sA Show Annotations.
 * @param vMag Vertical scale.
 * @param hMag Horizontal Scale.
 * @param newSample New start sample.
 * @throws TimeNotFoundException Throws exception when time is not found in record.
 */
  public void redraw(boolean sG, boolean sB, boolean sSN, boolean sA, float vMag, float hMag, int newSample) throws Tim
      Rectangle r = getBounds();
      int newSamp;
      newSamp = newSample;
      this.show_grid = sG;
      this.show_baselines = sB;
      this.show_signal_name = sSN;
      this.show_annotation = sA;
      this.vScale = vMag;
      this.hScale = hMag;

      // Check if the new sample number is in the record
      if (newSamp ≥ gprocess.getRecordLength())
         throw new TimeNotFoundException();
/*    // Check if the last sample point displayed is in the record and if
      // not, moves the screen to display the last portion of the record.
      if (newSamp > (gprocess.getRecordLength() - nSamples - 1))
         this.startSamp = gprocess.getRecordLength() - nSamples - 1;
      else if (newSamp < 0)
         this.startSamp = 0;
```



```
      else*/ this.startSamp = newSamp;
      repaint();
  }

/** Paints a light gray grid that scales according to the time and
 * amplitude scale. The spacing between vertical lines will always
 * represent 0.2 seconds. The spacing between horizontal lines will
 * always represent 0.5 mV.
 * @param g Graphics component to modify.
 */
  public void paintGrid(Graphics g) {
      Rectangle r = getBounds();
      // hlines and vlines are the number of horiz and vert lines
      float gheight = gprocess.getGridHeight(r) * vScale;
      float gwidth = gprocess.getGridWidth(r)/hScale;
      int hlines = (int) (r.height/gheight +0.5);
      int vlines = (int) (r.width/gwidth +0.5);

      g.setColor(Color.lightGray);
      for (int i=0; i ≤ hlines; i++) {
         g.drawLine(0, ((int) (i*gheight +0.5)), r.width, ((int)(i*gheight +0.5))); }
      for (int i=0; i ≤ vlines; i++) {
         g.drawLine(((int)(i*gwidth + 0.5)), 0, ((int) (i*gwidth + 0.5)), r.height); }
  }

/** This method paints the signal base lines to screen. It is only used for
 * developer testing purposes.
 * @param g Graphical component.
 */
  void paintBaseLines(Graphics g) {
      int base;
      Rectangle r = getBounds();

      g.setColor(Color.red);
      for (int i = 0; i < nSigs; i++) {
         base = getBaseLine(i);
         g.drawLine(0, base, r.width, base);
         //System.out.println(base for Signal +i+ : +base);
      }
  }

/** Paints all signals using processed data from GraphProcess. PaintSignal
 * paints only a screen's length of the record starting from the sample number
 * input.
 * @param g Graphics component.
 * @param startSamp The starting sample.
 */
  public void paintSignal(Graphics g, int startSamp) {
      int[] sigXPoints;
      int[] sigYPoints;
      Rectangle r = getBounds();
      int vSpace, base, numPoints;

      vSpace = r.height/nSigs;
      g.setColor(Color.blue);
```

```
    if (nSamples ≤ r.width)
        numPoints = nSamples;
    else numPoints = r.width;
    for (int x = 0; x < nSigs; x++) {
        base = getBaseLine(x);
        sigXPoints = gprocess.getSignalXComponents(x, startSamp, nSamples, r.width);
        sigYPoints = gprocess.getSignalYComponents(x, startSamp, nSamples, r.width, base, vScale);
        g.drawPolyline(sigXPoints, sigYPoints, numPoints);  // draws a signal
    }
  }

/** Paints the signal names above the signal baselines.  This method calls
 * graphProcess to get the names.
 * @param g Graphical component.
 */
  public void paintSignalName(Graphics g) {
    String sigName;
    int y;

    g.setColor(Color.black);
    for (int i = 0; i < nSigs; i++) {
        sigName = gprocess.getSignalName(i);
        y = getBaseLine(i) - 10;
        g.drawString(sigName, 5, y);
    }
  }

/** This method paints the start time and the end time at the bottom corners of
 * the canvas.  PaintTime() calls a graphProcess method to get the time in
 * the correct format.
 * @param g Graphical component.
 */
  public void paintTime(Graphics g) {
    Double sTime, eTime;
    String startTime, endTime;
    Rectangle r = getBounds();
    int x, y;
    y = r.height - 5;

    g.setColor(Color.black);
    startTime = gprocess.getTime(startSamp);
    endTime = gprocess.getTime(startSamp + nSamples);
    //startTime = sTime.toString();
    //endTime = eTime.toString();
    g.drawString(startTime, 5, r.height-5);
    g.drawString(endTime, r.width-75, r.height-5);
  }

/** Paints all annotations using processed data from graphProcess.  The text
 * annotations are always painted in the horizontal middle of the canvas and
 * correspoding vertical lines are drawn to more clearly denote the time of
 * the annotation.  Annotations are also painted based on the starting sample.
 *
 * @param g Graphical component.
 * @param startSamp Starting sample.
```



```
 */
  public void paintAnnotation(Graphics g, int startSamp) {
    Rectangle r = getBounds();
    Vector labels, positions = new Vector();
    Vector annotations;
    int position, yPlace;
    String label = "";
    int offset = 0;

    g.setColor(Color.red);
    annotations = gprocess.getAnnotations(startSamp, nSamples, r.width);
    positions = (Vector) annotations.elementAt(0);
    labels = (Vector) annotations.elementAt(1);
    yPlace = r.height/2 ;

    for (int i = 0; i < positions.size(); i++) {
        offset = 0;
        //for (Enumeration e = positions.elements(); e.hasMoreElements();) {
        position = ((Integer) positions.elementAt(i)).intValue();
        //System.out.println(position);
        label = (String) labels.elementAt(i);//.toString();
        if (label.equals("(N")) {
            offset = -15;}
        g.drawLine(position, 0, position, yPlace - 10 + offset);
        g.drawLine(position, yPlace + 10 + offset, position, r.height);
        g.drawString(label, position-3, yPlace+4 + offset);
    }
  }

/** This method takes an integer representing a signal to be shown on the canvas
 * and returns the baseline for that signal.
 * @param signal Signal number.
 * @return Baseline for signal.
 */
  public int getBaseLine(int signal) {
    Rectangle r = getBounds();
    int base;
    int vSpace = r.height/nSigs;

    //System.out.println(getBaseLine + r.height);
    base = vSpace*signal + vSpace/2;
    return base;
  }

}
```

```
/*
 * RecordControls.java
 *
 * Created on May 15, 2001, 3:54 PM
 * Created by Emily Liu
 */

package signalviewer;
import java.awt.event.*;
import javax.swing.*;

/** This class is responsible for all buttons and controls on the GUI.
 * The RecordControl class takes input from the RecordCanvas and
 * allows the user interaction with the display of the signals and
 * annotations. RecordControls contains all the controls for navigation
 * and search.
 */
public class RecordControls extends javax.swing.JPanel {
    // variables
    RecordCanvas canvas;
    boolean sG, sB, sSN, sA;
    float vMag, hMag;
    String searchAnnString;
    //int oneScreen; // in sample units

    // Constructor(s)
/** Constructs a new control panel. Record canvas initializes two sets of buttons:
 * The first is for navigation by time. The second set are forward and backward
 * search for annotations.
 * @param canvas Canvas component to modify.
 */
    public RecordControls(RecordCanvas canvas) {
        this.canvas = canvas;
        initComponents();
        BackSearchButton.setEnabled(sA);
        FowSearchButton.setEnabled(sA);
        //oneScreen = canvas.nSamples;
    }

    private void initComponents() {//GEN-BEGIN:initComponents
        BackOneButton = new javax.swing.JButton();
        BackHalfButton = new javax.swing.JButton();
        FindButton = new javax.swing.JButton();
        FowHalfButton = new javax.swing.JButton();
        FowOneButton = new javax.swing.JButton();
        BackSearchButton = new javax.swing.JButton();
        SearchAnnButton = new javax.swing.JButton();
        FowSearchButton = new javax.swing.JButton();
        setLayout(new java.awt.GridBagLayout());
        java.awt.GridBagConstraints gridBagConstraints1;
        setBorder(new javax.swing.border.EmptyBorder(new java.awt.Insets(1, 1, 1, 1)));

        BackOneButton.setLabel(" << ");
        BackOneButton.setBorder(new javax.swing.border.BevelBorder(0));
        BackOneButton.setText("  <<  ");
```



```
        BackOneButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                BackOneButtonActionPerformed(evt);
            }
        }
        );

        gridBagConstraints1 = new java.awt.GridBagConstraints();
        gridBagConstraints1.insets = new java.awt.Insets(0, 2, 0, 2);
        add(BackOneButton, gridBagConstraints1);


        BackHalfButton.setLabel(" < ");
        BackHalfButton.setBorder(new javax.swing.border.BevelBorder(0));
        BackHalfButton.setText("  <  ");
        BackHalfButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                BackHalfButtonActionPerformed(evt);
            }
        }
        );

        gridBagConstraints1 = new java.awt.GridBagConstraints();
        gridBagConstraints1.insets = new java.awt.Insets(0, 2, 0, 2);
        add(BackHalfButton, gridBagConstraints1);


        FindButton.setLabel(" Find ");
        FindButton.setBorder(new javax.swing.border.BevelBorder(0));
        FindButton.setText("  Go To  ");
        FindButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                FindButtonActionPerformed(evt);
            }
        }
        );

        gridBagConstraints1 = new java.awt.GridBagConstraints();
        gridBagConstraints1.insets = new java.awt.Insets(0, 2, 0, 2);
        add(FindButton, gridBagConstraints1);


        FowHalfButton.setLabel(" > ");
        FowHalfButton.setBorder(new javax.swing.border.BevelBorder(0));
        FowHalfButton.setText("  >  ");
        FowHalfButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                FowHalfButtonActionPerformed(evt);
            }
        }
        );

        gridBagConstraints1 = new java.awt.GridBagConstraints();
        gridBagConstraints1.insets = new java.awt.Insets(0, 2, 0, 2);
        add(FowHalfButton, gridBagConstraints1);
```

```
FowOneButton.setLabel(">>");
FowOneButton.setBorder(new javax.swing.border.BevelBorder(0));
FowOneButton.setText("  >>  ");
FowOneButton.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    FowOneButtonActionPerformed(evt);
  }
}
);

gridBagConstraints1 = new java.awt.GridBagConstraints();
gridBagConstraints1.insets = new java.awt.Insets(0, 2, 0, 30);
add(FowOneButton, gridBagConstraints1);


BackSearchButton.setLabel("<Search");
BackSearchButton.setBorder(new javax.swing.border.BevelBorder(0));
BackSearchButton.setText(" < Ann ");
BackSearchButton.setEnabled(false);
BackSearchButton.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    BackSearchButtonActionPerformed(evt);
  }
}
);

gridBagConstraints1 = new java.awt.GridBagConstraints();
gridBagConstraints1.insets = new java.awt.Insets(0, 30, 0, 2);
add(BackSearchButton, gridBagConstraints1);


SearchAnnButton.setBorder(new javax.swing.border.BevelBorder(0));
SearchAnnButton.setText("  Search...  ");
SearchAnnButton.setEnabled(false);
SearchAnnButton.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    SearchAnnButtonActionPerformed(evt);
  }
}
);

gridBagConstraints1 = new java.awt.GridBagConstraints();
gridBagConstraints1.insets = new java.awt.Insets(0, 2, 0, 2);
add(SearchAnnButton, gridBagConstraints1);


FowSearchButton.setLabel("Search>");
FowSearchButton.setBorder(new javax.swing.border.BevelBorder(0));
FowSearchButton.setText(" Ann > ");
FowSearchButton.setEnabled(false);
FowSearchButton.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(java.awt.event.ActionEvent evt) {
    FowSearchButtonActionPerformed(evt);
```



```
    }
  }
  );

  gridBagConstraints1 = new java.awt.GridBagConstraints();
  gridBagConstraints1.insets = new java.awt.Insets(0, 2, 0, 2);
  add(FowSearchButton, gridBagConstraints1);

}//GEN-END:initComponents

private void SearchAnnButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_SearchAnnButto
  JOptionPane jpane = new JOptionPane();
  Object input = jpane.showInputDialog(this.canvas, "Enter Annotation to Search:", "Annotation Search", JOptionPane.QU
  if (input != null) {
    searchAnnString = input.toString();
/*    try { canvas.redrawAnnotation(searchString, false);}
    catch (AnnotationNotFoundException e) {
      jpane.showMessageDialog(this.canvas, No Annotations Found!, Warning, JOptionPane.ERROR_MESSAGE);
    }*/
    BackSearchButton.setEnabled(true);
    FowSearchButton.setEnabled(true);
    jpane.showMessageDialog(this.canvas, "Annotation search set to: " + searchAnnString + ".  Please select forward or b
  }
}//GEN-LAST:event_SearchAnnButtonActionPerformed

private void BackSearchButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_BackSearchButt
  JOptionPane jpane = new JOptionPane();
/*  Object input = jpane.showInputDialog(this.canvas, Enter Annotation to Search:, Foward Annotation Search, JOptio
  if (input != null) {
    String searchString = input.toString();*/
    try { canvas.redrawAnnotation(searchAnnString, false);}
    catch (AnnotationNotFoundException e) {
      jpane.showMessageDialog(this.canvas, "No New Annotations Found!", "Warning", JOptionPane.ERROR_MESSAGE
    }
  //}
}//GEN-LAST:event_BackSearchButtonActionPerformed

private void FowSearchButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_FowSearchButto
// input:
// output:
// description:
  JOptionPane jpane = new JOptionPane();
/*  Object input = jpane.showInputDialog(this.canvas, Enter Annotation to Search:, Backward Annotation Search, JOpt
  if (input != null) {
    String searchString = input.toString();*/
    try { canvas.redrawAnnotation(searchAnnString, true);}
    catch (AnnotationNotFoundException e) {
      jpane.showMessageDialog(this.canvas, "No New Annotations Found!", "Warning", JOptionPane.ERROR_MESSAGE
    }
  //}
}//GEN-LAST:event_FowSearchButtonActionPerformed

private void FindButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_FindButtonActionPerf
// input:
// output:
```

```
// description:
/*    sG = canvas.show_grid;
    sB = canvas.show_baselines;
    sSN = canvas.show_signal_name;
    sA = canvas.show_annotation;*/
    JOptionPane jpane = new JOptionPane();
    Object input = jpane.showInputDialog(this.canvas, "Enter time in seconds:", "Goto Time", JOptionPane.QUESTION_ME
    if (input != null) {
        String inputValue = input.toString();
        try {
         canvas.redraw(inputValue);
        } catch (TimeFormatException e) {
            jpane.showMessageDialog(this.canvas, "Incorrect Time Format! Please enter again.", "Warning", JOptionPane.ERRC
            FindButtonActionPerformed(evt);
        } catch (TimeNotFoundException e) {
            jpane.showMessageDialog(this.canvas, "Time Not Found!", "Warning", JOptionPane.ERROR_MESSAGE);
        }
    }
}//GEN-LAST:event_FindButtonActionPerformed

private void BackOneButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_BackOneButtonAc
// input:
// output:
// description:
    sG = canvas.show_grid;
    sB = canvas.show_baselines;
    sSN = canvas.show_signal_name;
    sA = canvas.show_annotation;
    vMag = canvas.vScale;
    hMag = canvas.hScale;
    int moveSamp = canvas.startSamp - canvas.nSamples;
    try {
        canvas.redraw(sG, sB, sSN, sA, vMag, hMag, moveSamp);}
    catch (TimeNotFoundException e) {}
}//GEN-LAST:event_BackOneButtonActionPerformed

private void FowOneButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_FowOneButtonActi
// input:
// output:
// description:
    sG = canvas.show_grid;
    sB = canvas.show_baselines;
    sSN = canvas.show_signal_name;
    sA = canvas.show_annotation;
    vMag = canvas.vScale;
    hMag = canvas.hScale;
    int moveSamp = canvas.startSamp + canvas.nSamples;
    try {
        canvas.redraw(sG, sB, sSN, sA, vMag, hMag, moveSamp);}
    catch(TimeNotFoundException e) {}
}//GEN-LAST:event_FowOneButtonActionPerformed

private void BackHalfButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_BackHalfButtonAc
// input:
// output:
```



```
// description:
    sG = canvas.show_grid;
    sB = canvas.show_baselines;
    sSN = canvas.show_signal_name;
    sA = canvas.show_annotation;
    vMag = canvas.vScale;
    hMag = canvas.hScale;
    int moveSamp = canvas.startSamp - canvas.nSamples/2;
    try {
        canvas.redraw(sG, sB, sSN, sA, vMag, hMag, moveSamp);}
    catch (TimeNotFoundException e) {}
}//GEN-LAST:event_BackHalfButtonActionPerformed

private void FowHalfButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_FowHalfButtonActi
// input:
// output:
// description:
    int moveSamp = canvas.startSamp + canvas.nSamples/2;
    sG = canvas.show_grid;
    sB = canvas.show_baselines;
    sSN = canvas.show_signal_name;
    sA = canvas.show_annotation;
    vMag = canvas.vScale;
    hMag = canvas.hScale;
    try {
        canvas.redraw(sG, sB, sSN, sA, vMag, hMag, moveSamp);}
        catch (TimeNotFoundException e) {}
}//GEN-LAST:event_FowHalfButtonActionPerformed

/** This method will enable/disable the Annotation buttons based
 * on whether annotations are displayed to screen or not. If there
 * are no annotations loaded, the annotation buttons may not be used.
 * @param value User-specified annotation search value.
 */
public void setAnnotationButtons(boolean value) {
    if (value == false){
        BackSearchButton.setEnabled(value);
        FowSearchButton.setEnabled(value);
    }
    SearchAnnButton.setEnabled(value);
}

// Variables declaration - do not modify//GEN-BEGIN:variables
private javax.swing.JButton BackOneButton;
private javax.swing.JButton BackHalfButton;
private javax.swing.JButton FindButton;
private javax.swing.JButton FowHalfButton;
private javax.swing.JButton FowOneButton;
private javax.swing.JButton BackSearchButton;
private javax.swing.JButton SearchAnnButton;
private javax.swing.JButton FowSearchButton;
// End of variables declaration//GEN-END:variables

}
```

```
/*
 * recordObject.java
 *
 * Created on January 24, 2001, 5:39 PM
 */

package signalviewer;
import java.util.*;
import java.io.*;
import java.net.*;

/** This class is responsible for all graphical output on
 * the canvas. It will include a basic paint for grids, signals
 * and annotations.
 */
public class RecordObject extends Object {

  // Variables
/** The name of the record.
 *
 */
   public String recordName;
/** The sample frequency of the record.
 */
   public float sampFreq; // number of milliseconds between each sample
   private Vector time; // stores time in milliseconds
   private Vector signals; // list of SignalTypes
   private InputRecordStream inRecStrm;

   int numSignals;
   int numSamples;
   int done;          // 1 indicates all data has been retrieved
   int recordUpdate; // 1 indicates record is updating
   int knowSignals;  // 1 indicates recordObject knows number of signals

  // Contructor(s)
/** Constructs a record object and initializes the record data reading object.
 * InputRecordStream stores all the data within record. Record object then
 * stores the sampling frequency. Record Object also keeps the name of the
 * record and a list of the signals within the record.
 * @param inputString url string including variables concerning the record.
 */
   public RecordObject(String inputString) {
      System.out.println("Record Object with String reached");
      signals = new Vector();
      time = new Vector();
      inRecStrm = new InputRecordStream(inputString);
      inRecStrm.initSignals(this);
      inRecStrm.storeData(this);
      sampFreq = getSamplingFrequency();
   }

  // Methods
/** This method adds a new object representing a signal to the list that recordObject's
 * list of signals.
```



```
 * @param signal A signal object.
 */
   public void appendSignal(SignalType signal) {
      signals.add(signal);
   }

/** This method appends a new time to the time vector.
 * @param seconds Time in seconds.
 */
   public void appendTime(Double seconds) {
      time.add(seconds);
   }

/** This method returns the signalobject at index i.
 * @param i index of signal.
 * @return An object representing a signal.
 */
   public SignalType getSignalAt(int i) {
      return (SignalType) signals.elementAt(i);
   }

/** This method returns the requested data point of a given signal.
 * @param signal Signal index.
 * @param element Element of signal to access.
 * @throws ArrayIndexOutOfBoundsException Throws exception when there is no data at the specified
 * element of the specified signal.
 * @return Data at that point in the signal.
 */
   public Double getElementAt(int signal, int element) throws java.lang.ArrayIndexOutOfBoundsException{
      SignalType sT = getSignalAt(signal);
      return (Double) sT.getElementAt(element);
   }

   public Double getTimeAt(int i) {
      return (Double) time.elementAt(i);
   }

   public int getNumSignals() {
   // input:
   // output:
   // description: Returns the number of signals in this record
      return signals.size();
   }

   public float getSamplingFrequency() {
   // input:
   // output:
   // description: returns the sampling frequency
      //int x;

      //for (int i=0; i <
      //int one = getTimeAt(0).intValue();
      //int two = getTimeAt(1).intValue();
      float temp;
      int lastValue;
```

```
    int recordLength = getSignalLength(0);
    lastValue = getTimeAt(recordLength-1).intValue();
    temp = lastValue/(float)recordLength;
    System.out.println("Sampling Frequency: " + temp);
    return temp;
  }
  public boolean update() {
  // input:
  // output:
  // description:
    return true;
  }

  public boolean isComplete() {
  // input: none
  // output: boolean
  // description: returns 1 when all data in signals have been read
    return true;
  }

  public String getName() {
  // input: none
  // output: name
  // description: returns the name of the record
    return recordName;
  }

  public String getSignalName(int signal) {
  // input: none
  // output:
  // description:
    SignalType sT = getSignalAt(signal);
    String sigName = sT.getName();
    return sigName;
  }

  public int getSignalLength(int signal) {
  // input:
  // output
  // description:
    SignalType sT = getSignalAt(signal);
    return sT.getLength();
  }
  public void printTime() {
  // input: none
  // output: System output
  // description: Prints signals stored in record object. This method is
  //              provided as a testing tool.
    System.out.println("********** printing signalTime ***");
    for (Enumeration e = time.elements(); e.hasMoreElements();)
    {
      System.out.println(e.nextElement());
    }
  }// printTime
}
```

```
/*
 * SignalMenu.java
 *
 * Created on July 27, 2001, 12:17 AM
 */

package signalviewer;
import javax.swing.*;
import java.awt.*;

/** SignalMenu extends the javax.swing.JMenu.  This class is used as a bug
 * fix in order to show the menu when heavy and light components are mixed
 * in the GUI.  SignalMenu sets the JMenu to use heavy components instead of
 * light ones.
 * @author Emily Liu
 */
public class SignalMenu extends javax.swing.JMenuBar {

   /** Creates new SignalMenu */
   public SignalMenu() {
      JPopupMenu.setDefaultLightWeightPopupEnabled(false);
   }

}
```

```
/*
 * SignalType.java
 *
 * Created on June 1, 2001, 5:57 PM
 * purpose:  This class encapsulates the information about a single signal
 *           type and provides methods to manipulate data points in order
 *           to append and retrieve data.  It assumes that each signal has
 *           an associated time value in milliseconds.
 *
 */

package signalviewer;
import java.util.*;

public class SignalType extends java.lang.Object {
  // variables
   String signalName;
   Vector signalVector; // stores in microvolts

  // constructor
   public SignalType(String name) {
      signalName = name;
      signalVector = new Vector();
   }

  // methods
   public String getName() {
   // input: none
   // output: returns the name of the signal.
   // description:
      return signalName;
   }

   public void addElement(Double dPt) {
   // input: dPt
   // modifies: signal Vector
   // description:  appends the data point, dPt, to the end of signalVector
      signalVector.addElement (dPt);
   }

   public void add(int index, Double dPt) {
   // input: dPt
   // modifies: signal vector
   // description: inserts dPt at index
      signalVector.add(index, dPt);
   }

   public void add(Double dPt) {
   // input: dPt
   // modifies: signal vector
   // description: inserts dPt at index
      signalVector.add(dPt);
   }

   public Double getElementAt(int i) {
```



```
   // input: i
   // output: signalVector[i]
   // description:  returns the element at signalVector[i]
      return (Double) signalVector.elementAt(i);
   }

   public int getSignalSize() {
   // input: i
   // output:
   // description:
      return signalVector.size();
   }

   public void getSamples(int i, int n) {
   // input: i
   // output:
   // description:  Returns n samples begining from index i
   }

   public Double getStartPoint() {
   // input: none
   // output: the first point of the signal list
   // description:  returns the beginning point of the signal
      return (Double) signalVector.elementAt(0);
   }

   public int getLength() {
   // input: none
   // output: signal length
   // description: returns the number of points in the signal
      return signalVector.size();
   }

   public void printSignal() {
   // input: none
   // output: system output
   // description:
     System.out.println("signalName: " +signalName);
     for (Enumeration e = signalVector.elements(); e.hasMoreElements();) {
          System.out.println(e.nextElement());}
   }

   public String getInfo() {
   // input: none
   // output: none
   // description:  returns info about the class SignalType
      String info = "This class encapsulates the information about a signal type.";
      return info;
   }
}
```

```
package signalviewer;
import java.net.*;
import javax.swing.*;
import java.awt.*;

/** SignalViewer.java
 * This class will extend Applet and implement the highest level container for
 * a signal and annotation viewer.  The class will instantiate 4 other custom
 * classes in order to display a canvas and controls as well as read data and
 * annotation files.
 *
 * The Signal Viewer class is constructed by taken two arguments of the form:
 * <CODE>http://www.physionet.org/cgi-bin/rdsamp?database=___&record=___&start=___&end=___&doit=Show%2
 * <CODE>http://www.physionet.org/cgi-bin/rdann?database=___&record=___&annotator=___tstart=___&tend=___
 * @author Emily Liu
 * @since August 10, 2001
 */
public class SignalViewer extends javax.swing.JApplet {

  // Variables
  RecordCanvas canvas;
  RecordControls controls;
  GraphProcess gprocess;
  String recLocURL;
  String annLocURL;
  boolean sG, sB, sSN, sA;
  float vMag, hMag;

  // Constructors
/** This constructor currently has no function.
 */
  public SignalViewer() {
  } // end SignalViewer constructor

/** Init() initializes the applet.  This method constructs the GUI by creating
 * a menu bar, a RecordCanvas object, and a RecordControls object.  The init
 * method also initializes a intermediary GraphProcess object in order to access
 * record and annotation data.
 */
  public void init() {
    try{ recLocURL = getParameter("RecordURL");}
    catch (NullPointerException e) {
      System.out.println("RecLoc Param is null");
      //recLocURL=http://guest2:8082/classpath/signalviewer/mitdbrec100.htm;}
    }
    try{ annLocURL = getParameter("AnnURL");}
    catch (NullPointerException e) {
      System.out.println("AnnLoc Param is null");
      //annLocURL=;}
    }
      //annLocURL=http://guest2:8082/classpath/signalviewer/mitdbann100.htm;}
    if (annLocURL == null)
      gprocess = new GraphProcess(recLocURL);
    else gprocess = new GraphProcess(recLocURL, annLocURL);
    canvas = new RecordCanvas(gprocess);
```



```
    controls = new RecordControls(canvas);
    sG = canvas.show_grid;
    sB = canvas.show_baselines;
    sSN = canvas.show_signal_name;
    sA = canvas.show_annotation;
    vMag = canvas.vScale;
    hMag = canvas.hScale;
    getContentPane().setLayout(new BorderLayout());
    getContentPane().add(canvas, java.awt.BorderLayout.CENTER);
    getContentPane().add(controls, java.awt.BorderLayout.SOUTH);
    initComponents1();  //initializes the graphical components
    if (annLocURL == null){
      AnnotationCheckMenuItem.setEnabled(false);
      sA = false;
      controls.setAnnotationButtons(sA);
    }
  }
  // Methods
  private void initComponents1() {
    jMenuBar1 = new SignalMenu();
    searchMenu = new javax.swing.JMenu();
    FindTimeMenuItem = new javax.swing.JMenuItem();
    SearchForwardMenuItem = new javax.swing.JMenuItem();
    SearchBackwardMenuItem = new javax.swing.JMenuItem();
    VerticalMenu = new javax.swing.JMenu();
    ZeroVCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    FiftyVCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    HundredVCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    OneFiftyVCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    TwoHundredVCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    HorizontalMenu = new javax.swing.JMenu();
    ZeroHCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    FiftyHCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    HundredHCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    OneFiftyHCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    TwoHundredHCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    ViewMenu = new javax.swing.JMenu();
    GridCheckBoxMenuItem = new javax.swing.JCheckBoxMenuItem();
    AnnotationCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    SignalNameCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    helpMenu = new javax.swing.JMenu();
    helpMenuItem = new javax.swing.JMenuItem();
    jSeparator5 = new javax.swing.JSeparator();
    AboutMenuItem = new javax.swing.JMenuItem();

    /*searchMenu.setText(Search);

     FindTimeMenuItem.setText(Go to Time);
      FindTimeMenuItem.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
          FindTimeMenuItemActionPerformed(evt);
        }
      }
      );
      searchMenu.add(FindTimeMenuItem);
```

```
SearchForwardMenuItem.setText(Forward Annotation Search);
 SearchForwardMenuItem.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
       SearchForwardMenuItemActionPerformed(evt);
    }
 }
 );
 searchMenu.add(SearchForwardMenuItem);

SearchBackwardMenuItem.setText(Backward Annotation Search);
 SearchBackwardMenuItem.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
       SearchBackwardMenuItemActionPerformed(evt);
    }
 }
 );
 searchMenu.add(SearchBackwardMenuItem);
 jMenuBar1.add(searchMenu);*/

VerticalMenu.setText("Amplitude Scale");

 ZeroVCheckMenuItem.setText("0%");
  ZeroVCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
     public void actionPerformed(java.awt.event.ActionEvent evt) {
        ZeroVCheckMenuItemActionPerformed(evt);
     }
  }
  );
  VerticalMenu.add(ZeroVCheckMenuItem);

 FiftyVCheckMenuItem.setText("50%");
  FiftyVCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
     public void actionPerformed(java.awt.event.ActionEvent evt) {
        FiftyVCheckMenuItemActionPerformed(evt);
     }
  }
  );
  VerticalMenu.add(FiftyVCheckMenuItem);

 HundredVCheckMenuItem.setSelected(true);
  HundredVCheckMenuItem.setText("100%");
  HundredVCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
     public void actionPerformed(java.awt.event.ActionEvent evt) {
        HundredVCheckMenuItemActionPerformed(evt);
     }
  }
  );
  VerticalMenu.add(HundredVCheckMenuItem);

 OneFiftyVCheckMenuItem.setText("150%");
  OneFiftyVCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
     public void actionPerformed(java.awt.event.ActionEvent evt) {
        OneFiftyVCheckMenuItemActionPerformed(evt);
     }
```



```
     }
     }
     VerticalMenu.add(OneFiftyVCheckMenuItem);

    TwoHundredVCheckMenuItem.setText("200%");
     TwoHundredVCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
           TwoHundredVCheckMenuItemActionPerformed(evt);
        }
     }
     }
     VerticalMenu.add(TwoHundredVCheckMenuItem);
     jMenuBar1.add(VerticalMenu);

   HorizontalMenu.setText("Time Scale");

    ZeroHCheckMenuItem.setText("0%");
     ZeroHCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
           ZeroHCheckMenuItemActionPerformed(evt);
        }
     }
     }
     HorizontalMenu.add(ZeroHCheckMenuItem);

    FiftyHCheckMenuItem.setText("50%");
     FiftyHCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
           FiftyHCheckMenuItemActionPerformed(evt);
        }
     }
     }
     HorizontalMenu.add(FiftyHCheckMenuItem);

    HundredHCheckMenuItem.setSelected(true);
     HundredHCheckMenuItem.setText("100%");
     HundredHCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
           HundredHCheckMenuItemActionPerformed(evt);
        }
     }
     }
     HorizontalMenu.add(HundredHCheckMenuItem);

    OneFiftyHCheckMenuItem.setText("150%");
     OneFiftyHCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
           OneFiftyHCheckMenuItemActionPerformed(evt);
        }
     }
     }
     HorizontalMenu.add(OneFiftyHCheckMenuItem);

    TwoHundredHCheckMenuItem.setText("200%");
     TwoHundredHCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
```

```
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                TwoHundredHCheckMenuItemActionPerformed(evt);
            }
        }
        };
        HorizontalMenu.add(TwoHundredHCheckMenuItem);
        jMenuBar1.add(HorizontalMenu);

    ViewMenu.setText("View");

      GridCheckBoxMenuItem.setSelected(true);
        GridCheckBoxMenuItem.setText("Show Grid");
        GridCheckBoxMenuItem.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                GridCheckBoxMenuItemActionPerformed(evt);
            }
        }
        };
        ViewMenu.add(GridCheckBoxMenuItem);

      SignalNameCheckMenuItem.setSelected(true);
        SignalNameCheckMenuItem.setText("Show Signal Names");
        SignalNameCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                SignalNameCheckMenuItemActionPerformed(evt);
            }
        }
        };
        ViewMenu.add(SignalNameCheckMenuItem);

      AnnotationCheckMenuItem.setLabel("Show Annotations");
        AnnotationCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                AnnotationCheckMenuItemActionPerformed(evt);
            }
        }
        };
        ViewMenu.add(AnnotationCheckMenuItem);
        jMenuBar1.add(ViewMenu);

    helpMenu.setText("Help");

      helpMenuItem.setText("Help");
        helpMenu.add(helpMenuItem);

      helpMenu.add(jSeparator5);

      AboutMenuItem.setText("About...");
        helpMenu.add(AboutMenuItem);
        jMenuBar1.add(helpMenu);
      setName("Signal Viewer");
    setJMenuBar(jMenuBar1);
}
private void initComponents() {//GEN-BEGIN:initComponents
    jMenuBar1 = new javax.swing.JMenuBar();
```



```
    searchMenu = new javax.swing.JMenu();
    FindTimeMenuItem = new javax.swing.JMenuItem();
    SearchForwardMenuItem = new javax.swing.JMenuItem();
    SearchBackwardMenuItem = new javax.swing.JMenuItem();
    VerticalMenu = new javax.swing.JMenu();
    ZeroVCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    FiftyVCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    HundredVCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    OneFiftyVCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    TwoHundredVCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    HorizontalMenu = new javax.swing.JMenu();
    ZeroHCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    FiftyHCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    HundredHCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    OneFiftyHCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    TwoHundredHCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    ViewMenu = new javax.swing.JMenu();
    GridCheckBoxMenuItem = new javax.swing.JCheckBoxMenuItem();
    AnnotationCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    SignalNameCheckMenuItem = new javax.swing.JCheckBoxMenuItem();
    helpMenu = new javax.swing.JMenu();
    helpMenuItem = new javax.swing.JMenuItem();
    jSeparator5 = new javax.swing.JSeparator();
    AboutMenuItem = new javax.swing.JMenuItem();

    searchMenu.setText("Search");

      FindTimeMenuItem.setText("Go to Time");
        FindTimeMenuItem.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                FindTimeMenuItemActionPerformed(evt);
            }
        }
        };
        searchMenu.add(FindTimeMenuItem);

      SearchForwardMenuItem.setText("Forward Annotation Search");
        SearchForwardMenuItem.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                SearchForwardMenuItemActionPerformed(evt);
            }
        }
        };
        searchMenu.add(SearchForwardMenuItem);

      SearchBackwardMenuItem.setText("Backward Annotation Search");
        SearchBackwardMenuItem.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                SearchBackwardMenuItemActionPerformed(evt);
            }
        }
        };
        searchMenu.add(SearchBackwardMenuItem);
        jMenuBar1.add(searchMenu);
```

```
VerticalMenu.setText("Amplitude Scale");

ZeroVCheckMenuItem.setText("0%");
 ZeroVCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
       ZeroVCheckMenuItemActionPerformed(evt);
    }
 }
 );
 VerticalMenu.add(ZeroVCheckMenuItem);

FiftyVCheckMenuItem.setText("50%");
 FiftyVCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
       FiftyVCheckMenuItemActionPerformed(evt);
    }
 }
 );
 VerticalMenu.add(FiftyVCheckMenuItem);

HundredVCheckMenuItem.setSelected(true);
 HundredVCheckMenuItem.setText("100%");
 HundredVCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
       HundredVCheckMenuItemActionPerformed(evt);
    }
 }
 );
 VerticalMenu.add(HundredVCheckMenuItem);

OneFiftyVCheckMenuItem.setText("150%");
 OneFiftyVCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
       OneFiftyVCheckMenuItemActionPerformed(evt);
    }
 }
 );
 VerticalMenu.add(OneFiftyVCheckMenuItem);

TwoHundredVCheckMenuItem.setText("200%");
 TwoHundredVCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
       TwoHundredVCheckMenuItemActionPerformed(evt);
    }
 }
 );
 VerticalMenu.add(TwoHundredVCheckMenuItem);
 jMenuBar1.add(VerticalMenu);

HorizontalMenu.setText("Time Scale");

ZeroHCheckMenuItem.setText("0%");
 ZeroHCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
       ZeroHCheckMenuItemActionPerformed(evt);
```



```
    }
 }
 );
 HorizontalMenu.add(ZeroHCheckMenuItem);

FiftyHCheckMenuItem.setText("50%");
 FiftyHCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
       FiftyHCheckMenuItemActionPerformed(evt);
    }
 }
 );
 HorizontalMenu.add(FiftyHCheckMenuItem);

HundredHCheckMenuItem.setSelected(true);
 HundredHCheckMenuItem.setText("100%");
 HundredHCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
       HundredHCheckMenuItemActionPerformed(evt);
    }
 }
 );
 HorizontalMenu.add(HundredHCheckMenuItem);

OneFiftyHCheckMenuItem.setText("150%");
 OneFiftyHCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
       OneFiftyHCheckMenuItemActionPerformed(evt);
    }
 }
 );
 HorizontalMenu.add(OneFiftyHCheckMenuItem);

TwoHundredHCheckMenuItem.setText("200%");
 TwoHundredHCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
       TwoHundredHCheckMenuItemActionPerformed(evt);
    }
 }
 );
 HorizontalMenu.add(TwoHundredHCheckMenuItem);
 jMenuBar1.add(HorizontalMenu);

ViewMenu.setText("View");

GridCheckBoxMenuItem.setSelected(true);
 GridCheckBoxMenuItem.setText("Show Grid");
 GridCheckBoxMenuItem.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
       GridCheckBoxMenuItemActionPerformed(evt);
    }
 }
 );
 ViewMenu.add(GridCheckBoxMenuItem);
```

```
    AnnotationCheckMenuItem.setLabel("Show Annotations");
     AnnotationCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            AnnotationCheckMenuItemActionPerformed(evt);
        }
     }
     };
     ViewMenu.add(AnnotationCheckMenuItem);

    SignalNameCheckMenuItem.setSelected(true);
     SignalNameCheckMenuItem.setText("Show Signal Names");
     SignalNameCheckMenuItem.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            SignalNameCheckMenuItemActionPerformed(evt);
        }
     }
     };
     ViewMenu.add(SignalNameCheckMenuItem);
     jMenuBar1.add(ViewMenu);

   helpMenu.setText("Help");

    helpMenuItem.setText("Help");
     helpMenu.add(helpMenuItem);

    helpMenu.add(jSeparator5);

    AboutMenuItem.setText("About...");
     helpMenu.add(AboutMenuItem);
     jMenuBar1.add(helpMenu);
    setName("Signal Viewer");
   setJMenuBar(jMenuBar1);

 }//GEN-END:initComponents

private void TwoHundredHCheckMenuItemActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_Two
   vMag = canvas.vScale;
   hMag = 0.25f;
   canvas.redrawScale(vMag, hMag);
   ZeroHCheckMenuItem.setSelected(false);
   FiftyHCheckMenuItem.setSelected(false);
   HundredHCheckMenuItem.setSelected(false);
   OneFiftyHCheckMenuItem.setSelected(false);
}//GEN-LAST:event_TwoHundredHCheckMenuItemActionPerformed

private void ZeroHCheckMenuItemActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_ZeroHCheck
   vMag = canvas.vScale;
   hMag = 2.0f;
   canvas.redrawScale(vMag, hMag);
   FiftyHCheckMenuItem.setSelected(false);
   HundredHCheckMenuItem.setSelected(false);
   OneFiftyHCheckMenuItem.setSelected(false);
   TwoHundredHCheckMenuItem.setSelected(false);
}//GEN-LAST:event_ZeroHCheckMenuItemActionPerformed
```



```
private void TwoHundredVCheckMenuItemActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_Two
// description:
   vMag = 0.125f;
   hMag = canvas.hScale;
   canvas.redrawScale(vMag, hMag);
   ZeroVCheckMenuItem.setSelected(false);
   FiftyVCheckMenuItem.setSelected(false);
   OneFiftyVCheckMenuItem.setSelected(false);
   HundredVCheckMenuItem.setSelected(false);
}//GEN-LAST:event_TwoHundredVCheckMenuItemActionPerformed

private void ZeroVCheckMenuItemActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_ZeroVCheck
// description
   vMag = 0.013f;
   hMag = canvas.hScale;
   canvas.redrawScale(vMag, hMag);
   FiftyVCheckMenuItem.setSelected(false);
   HundredVCheckMenuItem.setSelected(false);
   OneFiftyVCheckMenuItem.setSelected(false);
   TwoHundredVCheckMenuItem.setSelected(false);
}//GEN-LAST:event_ZeroVCheckMenuItemActionPerformed

private void OneFiftyHCheckMenuItemActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_OneFifty
// description:
   vMag = canvas.vScale;
   hMag = 0.5f;
   canvas.redrawScale(vMag, hMag);
   ZeroHCheckMenuItem.setSelected(false);
   FiftyHCheckMenuItem.setSelected(false);
   HundredHCheckMenuItem.setSelected(false);
   TwoHundredHCheckMenuItem.setSelected(false);
}//GEN-LAST:event_OneFiftyHCheckMenuItemActionPerformed

private void HundredHCheckMenuItemActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_Hundred
// description:
   vMag = canvas.vScale;
   hMag = 1f;
   canvas.redrawScale(vMag, hMag);
   ZeroHCheckMenuItem.setSelected(false);
   FiftyHCheckMenuItem.setSelected(false);
   OneFiftyHCheckMenuItem.setSelected(false);
   TwoHundredHCheckMenuItem.setSelected(false);
}//GEN-LAST:event_HundredHCheckMenuItemActionPerformed

private void FiftyHCheckMenuItemActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_FiftyHCheck
// description:
   vMag = canvas.vScale;
   hMag = 1.5f;
   canvas.redrawScale(vMag, hMag);
   ZeroHCheckMenuItem.setSelected(false);
   HundredHCheckMenuItem.setSelected(false);
   OneFiftyHCheckMenuItem.setSelected(false);
   TwoHundredHCheckMenuItem.setSelected(false);
}//GEN-LAST:event_FiftyHCheckMenuItemActionPerformed
```

```
private void OneFiftyVCheckMenuItemActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_OneFifty
// description:
    vMag = 0.1f;
    hMag = canvas.hScale;
    canvas.redrawScale(vMag, hMag);
    ZeroVCheckMenuItem.setSelected(false);
    FiftyVCheckMenuItem.setSelected(false);
    HundredVCheckMenuItem.setSelected(false);
    TwoHundredVCheckMenuItem.setSelected(false);
}//GEN-LAST:event_OneFiftyVCheckMenuItemActionPerformed

private void HundredVCheckMenuItemActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_Hundred
// description:
    vMag = 0.05f;
    hMag = canvas.hScale;
    canvas.redrawScale(vMag, hMag);
    ZeroVCheckMenuItem.setSelected(false);
    FiftyVCheckMenuItem.setSelected(false);
    OneFiftyVCheckMenuItem.setSelected(false);
    TwoHundredVCheckMenuItem.setSelected(false);
}//GEN-LAST:event_HundredVCheckMenuItemActionPerformed

private void FiftyVCheckMenuItemActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_FiftyVCheck
// description:
    vMag = 0.025f;
    hMag = canvas.hScale;
    canvas.redrawScale(vMag, hMag);
    ZeroVCheckMenuItem.setSelected(false);
    HundredVCheckMenuItem.setSelected(false);
    OneFiftyVCheckMenuItem.setSelected(false);
    TwoHundredVCheckMenuItem.setSelected(false);
}//GEN-LAST:event_FiftyVCheckMenuItemActionPerformed

private void SearchBackwardMenuItemActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_SearchBa
// description:
    JOptionPane jpane = new JOptionPane();
    Object input = jpane.showInputDialog(this.canvas, "Enter Annotation to Search:", "Foward Annotation Search", JOptionP
    if (input != null) {
        String searchString = input.toString();
        try { canvas.redrawAnnotation(searchString, false);}
        catch (AnnotationNotFoundException e) {
            jpane.showMessageDialog(this.canvas, "No Annotations Found!", "Warning", JOptionPane.ERROR_MESSAGE);
        }
    }
}//GEN-LAST:event_SearchBackwardMenuItemActionPerformed

private void SearchForwardMenuItemActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_SearchForw
// description:
    JOptionPane jpane = new JOptionPane();
    Object input = jpane.showInputDialog(this.canvas, "Enter Annotation to Search:", "Backward Annotation Search", JOptio
    if (input != null) {
        String searchString = input.toString();
        try { canvas.redrawAnnotation(searchString, true);}
        catch (AnnotationNotFoundException e) {
            jpane.showMessageDialog(this.canvas, "No Annotations Found!", "Warning", JOptionPane.ERROR_MESSAGE);
```



```
        }
    }
}//GEN-LAST:event_SearchForwardMenuItemActionPerformed

private void FindTimeMenuItemActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_FindTimeMenu
// description:
    JOptionPane jpane = new JOptionPane();
    Object input = jpane.showInputDialog(this.canvas, "Enter time in seconds:", "Find Time", JOptionPane.QUESTION_MES
    if (input != null) {
        String inputValue = input.toString();
        try {canvas.redraw(inputValue);}
        catch (TimeFormatException e) {
            jpane.showMessageDialog(this.canvas, "Incorrect Time Format! Please enter again.", "Warning", JOptionPane.ERR
            FindTimeMenuItemActionPerformed(evt);
        }
        catch (TimeNotFoundException e) {
            jpane.showMessageDialog(this.canvas, "Time Not Found!", "Warning", JOptionPane.ERROR_MESSAGE);
        }
    }
}//GEN-LAST:event_FindTimeMenuItemActionPerformed

private void SignalNameCheckMenuItemActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_SignalN
// description:
    int startsamp = canvas.startSamp;
    if (sSN == true)
        sSN = false;
    else sSN = true;
    try {canvas.redraw(sG, sB, sSN, sA, vMag, hMag, startsamp);}
    catch(TimeNotFoundException e) {}
}//GEN-LAST:event_SignalNameCheckMenuItemActionPerformed

private void AnnotationCheckMenuItemActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_Annota
// description:
    int startsamp = canvas.startSamp;
    if (sA == true)
        sA = false;
    else sA = true;
    controls.setAnnotationButtons(sA);
    try {canvas.redraw(sG, sB, sSN, sA, vMag, hMag, startsamp);}
    catch(TimeNotFoundException e) {}
}//GEN-LAST:event_AnnotationCheckMenuItemActionPerformed

private void GridCheckBoxMenuItemActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_GridCheck
// description:
    int startsamp = canvas.startSamp;
    if (sG == true)
        sG = false;
    else sG = true;
    try {canvas.redraw(sG, sB, sSN, sA, vMag, hMag, startsamp);}
    catch (TimeNotFoundException e) {}
}//GEN-LAST:event_GridCheckBoxMenuItemActionPerformed

private void openMenuItemActionPerformed (java.awt.event.ActionEvent evt) {//GEN-FIRST:event_openMenuItemAct

}//GEN-LAST:event_openMenuItemActionPerformed
```

```
/** This method provides a description of the applet SignalViewer
 * @return A String description of the applet SignalViewer.
 */
 public String getLabel(){
    String label = "This class will extend Applet and implement the highest level container for a signal and annotion viewer. T
    return label;
 }

 // Variables declaration - do not modify//GEN-BEGIN:variables
 private javax.swing.JMenuBar jMenuBar1;
 private javax.swing.JMenu searchMenu;
 private javax.swing.JMenuItem FindTimeMenuItem;
 private javax.swing.JMenuItem SearchForwardMenuItem;
 private javax.swing.JMenuItem SearchBackwardMenuItem;
 private javax.swing.JMenu VerticalMenu;
 private javax.swing.JCheckBoxMenuItem ZeroVCheckMenuItem;
 private javax.swing.JCheckBoxMenuItem FiftyVCheckMenuItem;
 private javax.swing.JCheckBoxMenuItem HundredVCheckMenuItem;
 private javax.swing.JCheckBoxMenuItem OneFiftyVCheckMenuItem;
 private javax.swing.JCheckBoxMenuItem TwoHundredVCheckMenuItem;
 private javax.swing.JMenu HorizontalMenu;
 private javax.swing.JCheckBoxMenuItem ZeroHCheckMenuItem;
 private javax.swing.JCheckBoxMenuItem FiftyHCheckMenuItem;
 private javax.swing.JCheckBoxMenuItem HundredHCheckMenuItem;
 private javax.swing.JCheckBoxMenuItem OneFiftyHCheckMenuItem;
 private javax.swing.JCheckBoxMenuItem TwoHundredHCheckMenuItem;
 private javax.swing.JMenu ViewMenu;
 private javax.swing.JCheckBoxMenuItem GridCheckBoxMenuItem;
 private javax.swing.JCheckBoxMenuItem AnnotationCheckMenuItem;
 private javax.swing.JCheckBoxMenuItem SignalNameCheckMenuItem;
 private javax.swing.JMenu helpMenu;
 private javax.swing.JMenuItem helpMenuItem;
 private javax.swing.JSeparator jSeparator5;
 private javax.swing.JMenuItem AboutMenuItem;
 // End of variables declaration//GEN-END:variables


}
```

```
/*
 * TimeFormatException.java
 *
 * Created on July 30, 2001, 12:57 PM
 */

package signalviewer;

/** TimeFormatException is thrown whenever the time format is incorrect.
 */
public class TimeFormatException extends java.lang.Exception {

  /**
 * Creates new <code>TimeFormatException</code> without detail message.
  */
  public TimeFormatException() {
  }


  /**
 * Constructs an <code>TimeFormatException</code> with the specified detail message.
  * @param msg the detail message.
  */
  public TimeFormatException(String msg) {
    super(msg);
  }
}
```

```
/*
 * TimeNotFoundException.java
 *
 * Created on July 30, 2001, 1:43 PM
 */

package signalviewer;

/** TimeNotFoundException is thrown when the sample corresponding to
 * a time is not found within a record.
 */
public class TimeNotFoundException extends java.lang.Exception {

    /**
 * Creates new <code>TimeNotFoundException</code> without detail message.
    */
    public TimeNotFoundException() {
    }


    /**
 * Constructs an <code>TimeNotFoundException</code> with the specified detail message.
    * @param msg the detail message.
    */
    public TimeNotFoundException(String msg) {
        super(msg);
    }
}
```