

Remote Logging in Java using Jeli: A Facility to Enhance Development of Accessible Educational Software

Steven Robbins
Division of Computer Science
University of Texas at San Antonio
srobbins@cs.utsa.edu

Abstract

The combination of Java and the World Wide Web has opened up new opportunities for teaching at all levels. It is now possible to assume that all students in a class have access to the web through a browser that supports the Java language and a standard subset of the GUI API. One of the drawbacks of using Java through a browser is the lack of a standardized safe way for a Java applet to access resources on the local machine. Security measures prevent the applet from storing information generated by the applet on the local machine. The Jeli package contains a logging facility that allows an applet to store files either locally (if permitted) or on the server from which the applet was loaded. Jeli logging makes it significantly easier for instructors to develop applets that can permanently store information generated by user interaction with the application. The log can then be used by the student for study or the instructor for grading.

1 Introduction

Online learning is reshaping the vision of undergraduate education [2, 4]. Over the last three years, a large number of web-based instructional modules have been designed to supplement the undergraduate curriculum. Still, it has been estimated that fewer than one percent of all faculty will ever write a reusable lesson [7]. Converting an ad hoc piece of software into something that can be used by others is often difficult and tedious. One solution is to use modules that can easily be incorporated into other programs. We describe one such module that can be incorporated into any Java program to monitor or log the user interactions and program output.

Logging is important in any application in which data is gen-

erated in response to a user. An example of an application in which logging plays a critical role is interactive simulation. Interactive simulations allow students to perform experiments similarly to what was done a decade ago in a traditional laboratory. In a typical interactive simulation, the user provides input, performs experiments and analyzes the resulting data. Unless the output is particularly simple and can be copied from the screen, a facility to save the results of an experiment is desirable. This logging of data is often the most tedious part of the experimental process. A well-designed experiment will provide a semi-automatic logging facility so that the user can concentrate on the design of the experiment and the analysis of the data. An appropriately designed logging facility can also make the production of reports much easier.

The purpose of this paper is to describe part of a Java package called Jeli that allows developers to easily incorporate logging (storing files on disk) into both Java applets and applications with a minimal amount of difficulty. In the next section we discuss some desirable features for a logging system. We then discuss the limitations of logging in Java. A simple example of local logging using Jeli is presented followed by the extension to remote logging.

2 Desirable Features of a Logging System

Logging refers to the creation of a record of events. For example, in a scientific experiment an experimenter keeps a log book about experiments done. Data such as the input that the experiment needs is written down along with the procedures used so that the experiment can be recreated from this information. Often one does not know which experiments will be important, so all experiments should be logged. The simplest way to ensure that this will be done is to make the logging automatic and transparent.

When a student performs a simulation, the parameters are fed into the simulator and both the input and the results should be automatically logged. If the logging system is flexible enough, students can enter notes into the log file before or after the experiment. If done carefully the first time, the log file can be handed in as a report to be graded.

The purpose of a utility package to perform a function such as logging is to remove the necessity of rewriting a logging facility each time it is needed. A general purpose logging package should allow its incorporation into an existing or new program with a minimal amount of work.

A logging system will need to interact with the user in a number of ways. User interactions include:

- open the log
- close the log
- start the log
- stop the log
- display the log
- insert a comment into the log
- change the log filename

The simplest way to do this is to use buttons. Multifunction buttons allow for a simpler look and feel and use less screen real estate, often a limited resource. A single button can be used for opening and closing the log, with the label changing to `close log` when the log is open and back to `open log` when it is closed. The `start log`, `stop log` and `change log filename` can also share a single button, since the first two only make sense when the log is open and the last should be used only before opening the log file. Writing code that consistently handles the changing of the button labels is tedious and time consuming.

3 Java and Logging

The combination of Java and the World Wide Web has opened up new opportunities for teaching at all levels. It is now possible to assume that all students in a class have access to the web through a browser that supports the Java language and a standard subset of the GUI API. Java is available on just about every platform. Many free implementations are available along with a number of commercial ones. Machines are commonly purchased with a Java-capable browser installed.

While the standardization of Java and the GUI API is still in doubt, the popular browsers currently support a common, sufficiently powerful subset to approximate the *write once, run everywhere* promise of the Java language.

Most software for educational use consists of simple passive demonstrations that illustrate a single concept. While these can be useful, they often do not excite the student and there is no direct way to measure whether any knowledge or insight is being conveyed. Java can directly interact with the user and allows students to explore areas that otherwise would not be possible.

Java programs take one of two forms, the standalone application, and the applet. Standalone applications are more powerful, but require that a Java runtime system be installed. A Java standalone application has all of the security drawbacks of a program in any other language. A Java application can introduce a virus into the host system, so it is necessary to

trust that it came from a reliable source.

Java applets, on the other hand, run in a sandbox and have limited access to the machine they are running on. Unless the security features are disabled by the user (or there are bugs in the Java runtime environment), Java applets can be run safely as they cannot modify the host system. This is both a blessing and a curse. While it allows users to experiment with applets without worrying about damage to their machine, applets that cannot store information have limited usefulness.

Depending on the version of Java being run and the environment used for running the applet, different restrictions apply. For example, by default, a Java applet running under Sun's JDK 1.1 appletviewer can read from and write to the local hard disk, while the same applet running under Sun's JDK 1.2 appletviewer can, by default, read from the local disk but not write to it. The default action of most browsers is to not allow applets to either read from or write to the local disk. Users are rightly reluctant to disable this security feature, since it would leave their machines open to remote attacks. One way to handle input is to have the system provide a text box into which the user can paste a copy of the input file [8]. Logging requires output to the local machine, and while it might be possible to use a similar technique to pass a log file, log files are typically larger than could easily be put in a paste buffer.

Other attempts to implement educational software have either completely ignored the logging issue [3] or have opted to run as a Java application [1] so as to avoid the sandbox. Eventually, there may be some standardization to Java security that will allow users to confidently allow some applets to access parts of their local machine, but this remains to be seen.

4 Local Logging with Jeli

The Jeli package contains a `StandardLogButtons` class that creates button-like widgets as described in Section 2. This class keeps track of the opening and closing of the log and changes the labels on the buttons when required. A `Log Comment` button is only available when the log is open. Otherwise, the same button is used to toggle between `Replace Old Log` and `Append To Old Log`. The complete code for a simple applet that handles logging is shown in Figure 1. Figure 2 shows what is displayed when the log is closed and Figure 3 shows the same applet after the `Open Log` button has been pushed.

The four buttons created by `StandardLogButtons` are at the bottom, and a `Log Image and Count` button is at the top. The `RegisterLogButton` method in `StandardLogButtons` allows the `Log Image and Count` button to be automatically disabled when the log is closed or stopped and enabled when the log is open or started.

The `StandardLogButtons` constructor takes a number of

```

/* <applet code = "LogDemo" WIDTH = 300 HEIGHT = 225>
</applet>
*/

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import Jeli.*;
import Jeli.Logging.*;
import Jeli.Widgets.*;

public class LogDemo extends Applet
    implements ActionListener {

    private int remote_port = -1;
    private Button LogImage;
    private HelpManager hm;
    private StandardLogButtons slb;
    private String browser_start =
        "/opt/local/netscape/netscape";
    private String browser_dir =
        "file:/data1/srobbins/java/myjava/LogDemo/";
    private Image pic;
    private int count = 0;

    public void init() {
        Font hmf;

        hm = new Font("Serif",Font.PLAIN,18);
        hm = new HelpManager("Log Test",hmf,null);
        slb = new StandardLogButtons(hm,"new user","mylog.html",
            "", "A logging test", "version 1",
            false, remote_port);
        setup_layout();
        pic = getImage(getDocumentBase(),"javalo.gif");
    }

    public void paint(Graphics g) {
        int y;
        Insets insets;
        insets = getInsets();
        y = getInsets().top + LogImage.getBounds().height;
        g.drawImage(pic,0,y,this);
    }

    private void setup_layout() {
        Panel mybuttons = new Panel();
        setLayout(new BorderLayout());
        mybuttons.setLayout(new GridLayout(4,1));
        add("North",LogImage = new Button("Log Image and Count"));
        mybuttons.add(slb.GetOpenButton());
        mybuttons.add(slb.GetStartButton());
        mybuttons.add(slb.GetCommentButton());
        mybuttons.add(slb.GetShowRemoteLogButton());
        add("South",mybuttons);
        LogImage.addActionListener(this);
        LogImage.setEnabled(false);
        slb.RegisterLogButton(LogImage);
        slb.SetBrowser(browser_start,browser_dir);
    }

    public void actionPerformed (ActionEvent e) {
        count++;
        slb.LogString("This is count number "+count);
        slb.GetLogger().LogImage(pic,"A simple image",null);
    }
}

```

Figure 1: A simple applet which uses Jeli logging.

parameters. The first is a HelpManager that is used by other subpackages of the Jeli system. Here it is used just to set the font used by the buttons. A particularly large font is used so it will show up better in the figures. The next 5 parameters are strings used in creation and initialization of the log file. The first is a user name that is displayed at the beginning of

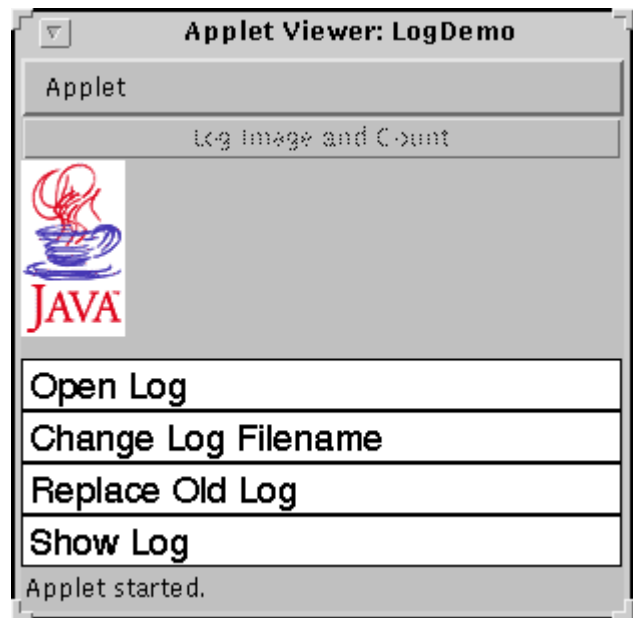


Figure 2: The demo window before the log file is open.

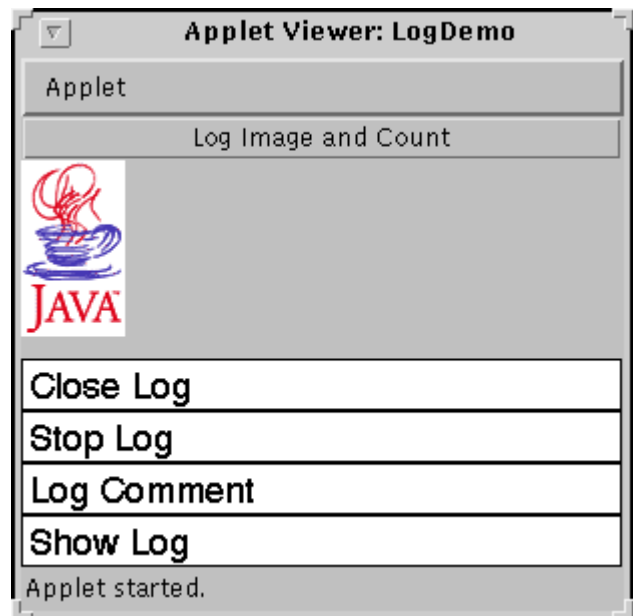


Figure 3: The demo window after the log file is open.

the log file. The second and third strings give the name of the file and the directory in which to store the file. The last two strings give a title and a version number that are displayed at the top of the log file.

The next parameter is a boolean variable that indicates whether to append to the old log file (true) or replace it (false). The last parameter is the remote port number described in the next section. Use 0 if not doing remote logging.

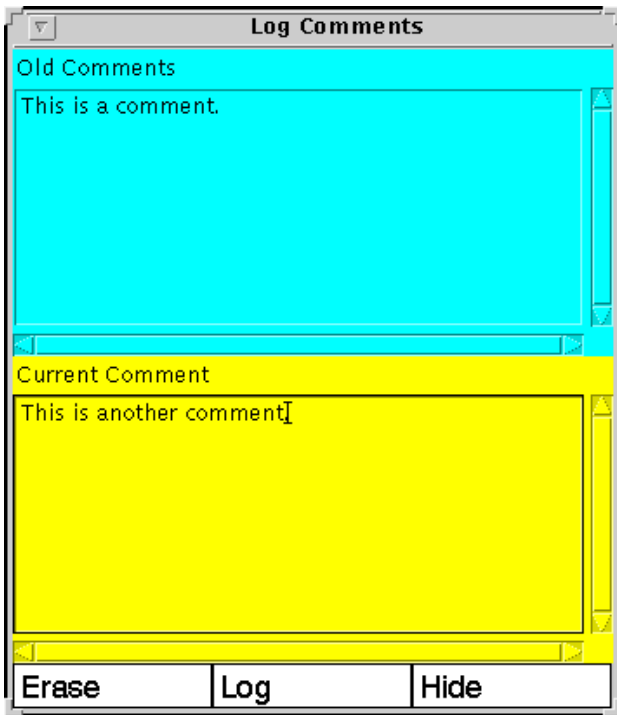


Figure 4: The frame for entering comments into the log file.

The logging facility is designed to create HTML files that can be viewed in a standard browser. The Show Log button can be used to pop up a browser showing the log file created. If local logging is used, the SetBrowser method sets the path to the browser to bring up.

Pushing the Log Comment button pops up a frame that allows the entry of comment into the log file. An example is shown in Figure 4. Comments are typed into the lower window of the frame. They can be edited until the Log button is pushed. At this time the Current Comment is put into the log file and saved in the Old Comments window. Figure 5 shows a simple log file created by opening the log file, logging an image, entering a comment, logging the image again and closing the log.

The Log Image and Count button is not part of the Jeli package and is included to illustrate how a program can insert information into the log file. By registering this button with the StandardLogButtons class, it can be automatically enabled and disabled as the logging is started and stopped. To simplify the user code, the LogString method of the package silently does nothing when logging is inactive.

The Log Image and Count button increments a variable that counts the number of times the button has been pushed. It inserts a string in the log file containing that count, and then inserts an image into the log file. The image used by the demo was read in from a GIF file when the applet was initialized, but it could be any Java image. Other parts of the

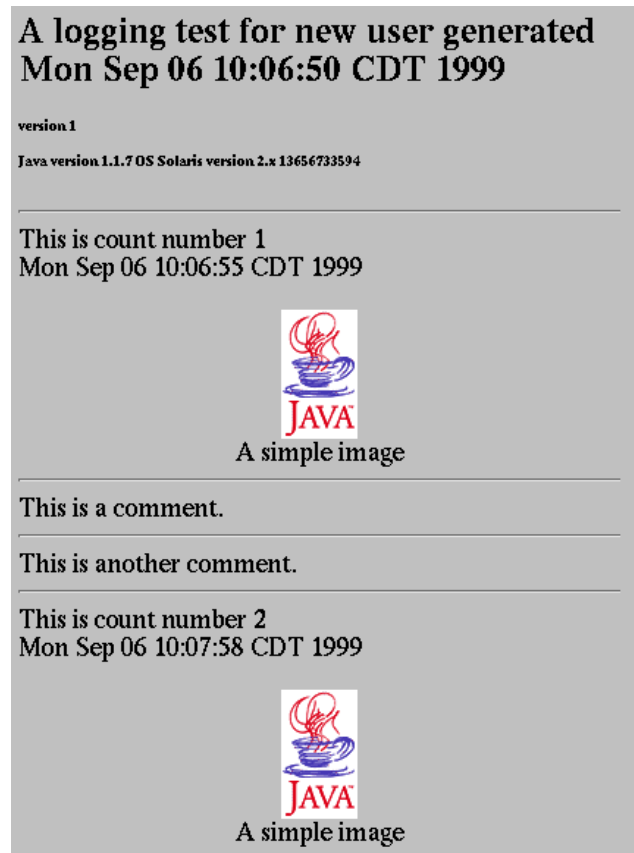


Figure 5: A sample log file created by the simple applet.

Jeli package use the logging of images to store graphs in the log file. When an image is logged, Jeli converts the image into a GIF file, stores it in the same directory as the log file, and inserts a link to the GIF file into the log. Each time a new image is logged, a unique filename is created for the GIF file.

5 Remote Logging

Java applets run from a browser usually may not access the local disk. This behavior is desirable, or the simple action of clicking on a link in a browser could allow a hostile applet to write to your disk.

Java applets may initiate a network connection back to the machine from which the applet was loaded. Jeli contains a remote logging server that allows an applet to use this type of connection to store files in a given directory on the host. Precautions are taken so that a remote applet cannot fill up the disk on the server machine.

The server is a Java application that takes the following command line parameters.

- The base directory for storing the files. The server stores files in a directory structure under the directory in which it was started. If this directory is accessible

by a web server, the base directory parameter should be the web address used to get to this directory. This parameter is passed back to the client, so that it knows the web address for viewing the log files.

- The port number for connecting to the server.
- The maximum number of megabytes that may be written to the disk for all connections to the server.
- The maximum number of megabytes that may be written during one connection to the server.

The server accepts connections from the Jeli logging facility. Connection requests have a particular format that makes it unlikely that an accidental request from another source will influence the contents of the disk on the server. The server allows only for creating files, not for reading or deleting them. The only access to the files is through a browser. Each connection creates files in a new directory. The name of the directory is based on the user name passed to the `StandardLogButtons` constructor, but it is guaranteed to be unique so that the same user can perform multiple experiments and keep them separate. It is expected that the server will be started from a directory accessible to a web server running on the same machine. The URL of the created directory is sent back to the logging package so that pushing the `Show Log` button allows web pages in that directory to be automatically displayed in a standard browser.

The server keeps track of the number of bytes logged and refuses additional connections and logging requests if the specified limits are exceeded. The limits prevent the logging facility from filling up the disk on the web server, but they still allow a malicious user to prevent others from using the logging system.

The only change needed in the code in Figure 1 to use remote logging is to specify a non-zero port number in the `StandardLogButtons` constructor. This port number must be the same as the one used to start the server.

The remote logging has several advantages. The program can be run from a browser and still produce log output. The log file can be read from a standard browser and therefore easily printed on the local machine.

The developer of the software can have others use it and create log files without the need for the user to download or install any software beyond a standard browser.

An instructor can install the software in an account owned by the instructor on a department machine and have the students create log files on that machines. These log files, once created, cannot be erased or modified by the student and so they give a valid log of what the student has done.

6 Availability

A number of interactive simulations that use the Jeli remote logging facility have been described in the literature [5, 6] and are available for use either remotely or by downloading the various Java packages [9]. The Jeli package that includes the logging facility described here is also available separately [10] for incorporation into other programs.

7 Acknowledgments

This work has been supported by NSF grants: *An Electronic Laboratory for Operating Systems and Computer Networks*, DUE-9750953 and *A Web-Based Electronic Laboratory for Operating Systems and Computer Networks*, DUE-9752165. Richard Fellingner implemented the remote logging feature of Jeli described here.

References

- [1] Barnett, L., et al., "Design and implementation of an interactive tutorial framework." *Proc. 29th SIGCSE Technical Symposium on Computer Science Education*, 1998, pp. 87-91.
- [2] Harasim, L., et al., *Learning Networks: A Field Guide to Teaching and Learning Online*, the MIT Press, Cambridge, Mass., 1995.
- [3] Khuri, S. and Hsiu-Chin, H., "Visualizing the CPU scheduler and page replacement algorithms," *Proc. 30th SIGCSE Technical Symposium on Computer Science Education*, 1999, pp. 227-231.
- [4] Hechinger, J., "Textbook publisher lays plans for an Internet university," *The Wall Street Journal*, July 2, 1999.
- [5] Robbins, S. and Robbins, K., "Empirical exploration in undergraduate operating systems," *Proc. 30th SIGCSE Technical Symposium on Computer Science Education*, 1999, pp. 311-315.
- [6] Robbins, S., "Experimentation with bounded buffer synchronization," *Proc. 31st SIGCSE Technical Symposium on Computer Science Education*, 2000.
- [7] Roschelle, J., et al., "Developing educational software components," *Computer*, September, 1999, pp. 50-58.
- [8] Stasko, J., JSamba, a Java version of Samba, described at <http://www.cc.gatech.edu/gvu/softviz/SoftViz.html>
- [9] <http://vip.cs.utsa.edu/nsf/>
- [10] <http://vip.cs.utsa.edu/nsf/Jeli.html>