Log

Contents

1 Howto log

A Logger object is added to the main Class Nectar. We must add it because if we make our own class, the logging messages will display this new class instead of the original calling line.

`UI_Log` and `GUI_Log` will both log into the log file and the log window if it is allocated.

```
package nectar;
import java.io.IOException;
import java.util.logging.FileHandler;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.logging.SimpleFormatter;

public class Nectar {
    Logger logger;

    Nectar() {
        // This block configure the logger with handler and formatter
        logger = Logger.getLogger("Nectar");
        FileHandler fh;

        try {
            fh = new FileHandler("/tmp/NectarFile.log", true);
            logger.addHandler(fh);
            logger.setLevel(Level.ALL);
            SimpleFormatter formatter = new SimpleFormatter();
            fh.setFormatter(formatter);

        } catch (SecurityException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        Nectar all = new Nectar();
        all.logger.log(Level.SEVERE,"testing SEVERE log level");
        all.logger.log(Level.WARNING,"testing WARNING log level");
        all.logger.log(Level.INFO,"testing INFO log level");
        all.logger.log(Level.CONFIG,"testing CONFIG log level");
        all.logger.log(Level.FINE,"testing FINE log level");
        all.logger.log(Level.FINER,"testing FINER log level");
        all.logger.log(Level.FINEST ,"testing FINEST log level");
    }
}
```

Note that you cannot log to a file using a non-signed applet.

2 Modify format

```
    /**
    * MyCustomFormatter formats the LogRecord as follows:
    * date   level   localized message with parameters
    */
    static public class MyCustomFormatter extends Formatter {

public MyCustomFormatter() {
            super();
}

public String format(LogRecord record) {
            // Create a StringBuffer to contain the formatted record
            // start with the date.
            StringBuffer sb = new StringBuffer();

            // Get the date from the LogRecord and add it to the buffer
            /*Date date = new Date(record.getMillis());
            sb.append(date.toString());
            sb.append(" ");*/

            // Get the calling function
            sb.append(record.getSourceClassName() + ".");
            sb.append(record.getSourceMethodName() + ": ");

            // Get the level name and add it to the buffer
            sb.append(record.getLevel().getName());
            sb.append("\t");

            // Get the formatted message (includes localization
            // and substitution of paramters) and add it to the buffer
            sb.append(formatMessage(record));
            sb.append("\n");

            return sb.toString();
}
    }
...
    MyCustomFormatter f = new MyCustomFormatter();
    FileHandler       fh = new FileHandler(logFile, true);
    fh.setFormatter(f);
    logger.addHandler(fh);
```

3 Log into a windows

```
    static class MyWindowHandler extends Handler {
        /**
        * This is the overridden publish method of the abstract super class
        * Handler. This method writes the logging information to the associated
        * Java window. This method is synchronized to make it thread-safe. In case
        * there is a problem, it reports the problem with the ErrorManager, only
        * once and silently ignores the others.
        *
        * @record the LogRecord object
```

```
        *
        */
        public synchronized void publish(LogRecord record) {
            String message = null;
            //check if the record is loggable
            if (!isLoggable(record))
                return;
            try {
                message = getFormatter().format(record);
                System.out.println(":) " + message);
            } catch (Exception e) {
                System.out.println("error: " + e.getMessage());
            }
        }

        public void close() {}
        public void flush() {}
    }
...
    MyWindowHandler wh = new MyWindowHandler();
    logger.addHandler(wh);
```