## CS 580 Client-Server Programming

### Fall Semester, 2002

#### Doc 18 Logging

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logging</td>
<td>2</td>
</tr>
<tr>
<td>Why logging?</td>
<td>3</td>
</tr>
<tr>
<td>Log File Examples</td>
<td>5</td>
</tr>
<tr>
<td>Logging Systems</td>
<td>11</td>
</tr>
<tr>
<td>VisualWorks – LoggingTool</td>
<td>11</td>
</tr>
<tr>
<td>LogEnvironment Logging Levels</td>
<td>14</td>
</tr>
<tr>
<td>LoggingTool Window</td>
<td>16</td>
</tr>
<tr>
<td>JDK 1.4 Logging</td>
<td>19</td>
</tr>
<tr>
<td>Log Levels</td>
<td>20</td>
</tr>
<tr>
<td>SimpleExample</td>
<td>21</td>
</tr>
<tr>
<td>Example of Different Message Types</td>
<td>24</td>
</tr>
<tr>
<td>Logger Names</td>
<td>29</td>
</tr>
<tr>
<td>Sample Configuration File</td>
<td>31</td>
</tr>
<tr>
<td>SDSU Logger</td>
<td>33</td>
</tr>
<tr>
<td>SDSU Logger Example</td>
<td>34</td>
</tr>
</tbody>
</table>

### References

- Java Logging Overview, [http://java.sun.com/j2se/1.4.1/docs/guide/util/logging/overview.html](http://java.sun.com/j2se/1.4.1/docs/guide/util/logging/overview.html)
- Java Logging API [http://java.sun.com/j2se/1.4.1/docs/api/java/util/logging/package-summary.html](http://java.sun.com/j2se/1.4.1/docs/api/java/util/logging/package-summary.html)
- Previous SDSU Client-Server Programming lecture notes
- Patterns for Logging Diagnostic Messages by Neil Harrison in *Pattern Languages of Program Design* 3 Eds Martin, Riehle, Buschman, 1998, pp 277-289

### Copyright

©, All rights reserved. 2002 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/opl.shtml) license defines the copyright on this document.
Logging

public void run()
{
    while (true)
    {
        try
        {
            Socket client = acceptor.accept();
            processRequest( client );
        }
        catch (IOException acceptError)
        {
            // for a later lecture - this is the lecture
        }
    }
}
Why logging?

- Performance tuning
- Upgrade justification
- Problem tracking
- Access counting

How many times was the assignment 1 server accessed?
What should be logged?

Normally a log entry contains several pieces of information:

- Date and time
- Service that caused the entry
- Client address that caused the entry
- Host on which the server runs
- Event
Log File Examples
Assignment 1 Log file

68.7.98.199: ?webSite POST /cs580 -> 200 text/xml 2002-10-19 22:24:47.0000
68.7.98.199: ?webSite POST /cs580 -> 200 text/xml 2002-10-19 22:24:47.0000

Apache
Access

/scripts/..%25%35%63../winnt/system32/cmd.exe/?c+dir HTTP/1.0" 404 303
211.90.88.43 - - [21/Oct/2002:08:33:30 -0700] "GET
/scripts/..%252f../winnt/system32/cmd.exe/?c+dir HTTP/1.0" 404 303

Error

[Mon Oct 21 08:33:29 2002] [error] [client 211.90.88.43] File does not exist: /opt/etc/apache-1.3.26/htdocs/scripts/..%5c../winnt/system32/cmd.exe
[Mon Oct 21 08:33:30 2002] [error] [client 211.90.88.43] File does not exist: /opt/etc/apache-1.3.26/htdocs/scripts/..%2f../winnt/system32/cmd.exe

Note that the log files contain the client IP address not the name of the machine
Dnews – A News Server

Log files

• dnews.in
• dnews.log
• dnews.out
• dnews_post.log
• expire.log
• status.log
• users.log

Configuration files

• access.conf
• control.conf
• expire.conf
• fts.conf
• hp.conf
• moderators.conf
• newsfeeds.conf
Dnews Log file examples

users.log

Mon Oct 21 04:50:23 Bytes Read, Current Connections, Area/IPrange
Mon Oct 21 04:50:23 197k 1 194.170.42.68
Mon Oct 21 04:50:23 197k 1 byu
Mon Oct 21 04:50:23 0mb 2 Grand Totals

dnews.log

21 09:41:27 :info: db_piles_trim called
21 09:42:08 :info: db_pile_status
21 09:42:08 :info: db_pile_status_done
21 09:42:08 :info: dnews_status 1
21 09:42:08 :info: dnews_status 2
21 09:42:08 :info: dnews_status 3
21 09:42:08 :info: Timezone Offset 25200 dst 3600
21 09:42:16 :info: Writing list of read groups to a file for this slave
21 09:42:16 :info: Wrote 17 records
21 09:42:28 :info: db_piles_trim called
Log file formats

Log files get big

Assignment1 log file was 4MB

Web server logs get 10 - 100 Mbs before rotating the file

Make the log file machine parsable!
How should clients and servers log?

Basic choices:

• Appending to a log file
• System logging facility
Simple appending problems

This simplistic approach can cause problems with the following:

• Concurrency

  Two server threads concurrently writing to the same logfile

• Performance

  Every log entry requires lots of overhead
  (opening and closing the logfile)

Some solutions:

• Rely on the OS to perform atomic appends to the log file. (One unbuffered write per log entry)

• Use a dedicated logging thread (Keep the log file open)

• Synchronize the writing to the log file.
Logging Systems
VisualWorks – LoggingTool

Load parcel LoggingTool in parcels directory

Mainly used for debugging, but will work for our purposes

The logging tool is in the namespace Tools

If your code is a namespace other than Smalltalk you either need to:

• Import “Tools” or “Smalltalk” in your classes or

• Import the “Tools” or “Smalltalk” in your namespace

• Use the full name Tools.LogEnvironment
Basic Operations

Accessing the logger

LogEnvironment default

Returns a unique instance of LogEnvironment

Turn logging on

LogEnvironment default traceOn.

Turn logging off

LogEnvironment default traceOff.

Logging

LogEnvironment default log: 'Here is a real meaningless log entry'.

Logging with levels

LogEnvironment default
  log: 'Out of memory error occurred'
  level: #error
Logging & Levels

Most logging systems have different levels like:

- Error
- Warning
- Debug

Source code logs messages at multiple levels

When you don’t need output of a log level

- Tell logging system to ignore a level
- Keep logging statements in the source code

When need that log level again

- Tell logging system to record that level
LogEnvironment Logging Levels

Levels indicated by symbols

Default level is #general

To turn logging of a level on:

   LogEnvironment default addDebugCategory: #error

To turn logging of a level on:

   LogEnvironment default removeDebugCategory: #error
LogEnvironment & The logs

Writes logs to a stream

Until the stream is set, all log requests are ignored

This stream is set by LoggingTool window

- Is set to write logs to a string
- Is set when you open the LoggingTool window

If you want logs to go to a file use

```
| logFile |
LogFile := 'server.log' asFilename.
LogEnvironment default debugStream: logFile writeStream
```

If you do the above you cannot use the LogginTool Window
LoggingTool Window

The VisualWorks logging system is mainly for debugging.

Log entries can be displayed in a Window.

To open the logging window:

  LoggingTool open

Your classes can register with LoggingTool to

  • Add menus to turn logging levels off & on
How to Register with LoggingTool

Smalltalk defineClass: #SampleClassWithLogging

SampleClassWithLogging class methods

menuItemName
   ^'SampleClass Messages'

dbLabelsAndValues
   ^(List new)
       add: 'Errors' -> #error;
       add: 'SampleClass Debug' -> #SampleClassDebug;
       add: 'SampleClass Commands' -> #SampleClassCommands;
       yourself

registerToDebug
   LogEnvironment default addToDebug: self

initialize
   self registerToDebug

log: aString level: aSymbol
   LogEnvironment default
       log: aString
       level: aSymbol
Sample Use of Logging Methods

SampleClassWithLogging methodsFor: 'method with logging '

foo
    [1 / 0] on: Exception
do:
        [:error |
            self class
                log: Timestamp now printString,
                    '; Error - SampleClassWithLogging>>foo; ',
                    error description
                level: #error].

self class
    log: Timestamp now printString, ' Sample Command '
    level: #SampleClassDebug
JDK 1.4 Logging

Starting with JDK 1.4 Java API has a logging system

Supports
• Multiple log levels
• Multiple output formats
• Output to different IO devices

• Filters for additional filtering of message to accept
• ResourceBundles for localization of log messages
• Initialization of loggers by configuration file
• Hierarchical loggers in one program
Log Levels

- ALL
- SEVERE (highest value)
- WARNING
- INFO (usual default)
- CONFIG
- FINE
- FINER
- FINEST (lowest value)
- OFF

Output formats

- XML (default for files output)
- Normal Text (default for screen output)

Output devices

- Stream
- System.err
- File or rotating set of files
- Socket for network logging
- Memory
SimpleExample

```java
import java.util.logging.*;

public class SimpleLoggingExample {
    private static Logger logger = Logger.getLogger("edu.sdsu.cs580");

    public static void main (String args[]) {
        new SimpleLoggingExample().someLogMessages();
    }

    public void someLogMessages() {
        logger.severe("A severe log message");
        logger.getLogger("edu.sdsu.cs580").fine("A fine message");
        logger.warning("Be careful");
    }
}
```

**Output To System.err**

SEVERE: A severe log message
WARNING: Be careful

**Default Settings**

Use a ConsoleHandler
Level set to INFO
System administrator can change default settings
Five Categories of Logging Messages

Convenience Methods

- severe( String message);
- warning( String message);
- info( String message);
- config( String message);
- fine( String message);
- finer( String message);
- finest( String message);

Convenience Methods for Tracing Methods

entering(String sourceClass, String sourceMethod);
entering(String sourceClass, String sourceMethod, Object parameter);
entering(String sourceClass, String sourceMethod, Object[] parameters);
exiting(String sourceClass, String sourceMethod);
exiting(String sourceClass, String sourceMethod, Object result);

Log Methods

log(Level logLevel, String message);
log(Level logLevel, String message, Object parameter);
log(Level logLevel, String message, Object[] parameters);
log(Level logLevel, String message, Throwable exception);

Currently parameters argument is ignored
Precise Log Methods

Add class and method to log messages

logp(Level logLevel, String class, String method, String message);

etc.

Logs with Resource Bundles

Add resource bundle to logp messages

logrb(Level logLevel, String class, String method, String bundlename, String message);

cetc.
Example of Different Message Types

```java
import java.io.*;
import java.util.Vector;
import java.util.logging.*;

public class MessageTypes
{
    private static Logger logger = Logger.getLogger("edu.sdsu.cs580");

    static
    {
        try
        {
            Handler textLog = new FileHandler("textLog.txt");
            textLog.setFormatter( new SimpleFormatter());
            textLog.setLevel(Level.ALL);
            Handler xmlLog = new FileHandler("xmlLog.txt");
            xmlLog.setFormatter( new XMLFormatter());
            xmlLog.setLevel(Level.ALL);
            logger.addHandler(textLog);
            logger.addHandler(xmlLog);
            logger.setLevel(Level.ALL);
        }
        catch (IOException fileError)
        {
            System.err.println( "Could not open log files");
        }
    }
```
Example Continued

public static void main (String args[]) {
    new MessageTypes().someLogMessages();
}

public void someLogMessages() {
    logger.entering("MessageTypes", "someLogMessages");
    Vector data = new Vector();
    data.add("Cat");
    logger.log(Level.SEVERE, "Show Vector", data);
    logger.severe("A severe log message");
    logger.logp(Level.SEVERE, "MessageTypes", "someLogMessages", "Logp example");
    try {
        int zeroDivide = 1 / (1 - 1);
    }
    catch (Exception zeroDivide) {
        logger.log(Level.SEVERE, "Exception example", zeroDivide);
    }
    logger.exiting("MessageTypes", "someLogMessages");
}
SimpleFormatter Output

FINER: ENTRY
Oct 22, 2002 11:56:42 AM MessageTypes someLogMessages
SEVERE: Show Vector
Oct 22, 2002 11:56:42 AM MessageTypes someLogMessages
SEVERE: A severe log message
Oct 22, 2002 11:56:42 AM MessageTypes someLogMessages
SEVERE: Logp example
Oct 22, 2002 11:56:42 AM MessageTypes someLogMessages
SEVERE: Exception example
java.lang.ArithmeticException: / by zero
   at MessageTypes.someLogMessages(MessageTypes.java:46)
   at MessageTypes.main(MessageTypes.java:34)
Oct 22, 2002 11:56:42 AM MessageTypes someLogMessages
FINER: RETURN

XMLFormatter Sample Output

<?xml version="1.0" encoding="US-ASCII" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
<record>
  <date>2002-10-22T11:56:41</date>
  <millis>1035313001981</millis>
  <sequence>0</sequence>
  <logger>edu.sdsu.cs580</logger>
  <level>FINER</level>
  <class>MessageTypes</class>
  <method>someLogMessages</method>
  <thread>10</thread>
  <message>ENTRY</message>
</record>
**FileHandlers**

Can be set to rotate files

Can be located in temp directory

Can be set to
- Append existing files
- Overwrite existing files (default)

To change append setting either

- Use constructor

  `FileHandler(String pattern, boolean append)`

- Or use configuration file
Loggers

Can have

• Multiple handlers

• Multiple handlers of same type

Loggers and handlers have differ log levels

Logger
• Drops all messages below it log level
• Passes remaining messages to all handlers
• Handler can further drop more messages
Logger Names

Logger names are arbitrary

Logger.getLogger("edu.sdsu.cs580")
Logger.getLogger("foo")
Logger.getLogger("")

Sun recommends using hierarchical names with format

"domain.package"
"domain.package.class"

Loggers inherit settings from “parent” logger

Logger "edu.sdsu.cs580" would inherit settings of "edu.sdsu"
Logger Scope

Logger settings can be defined in

- Program
- Configuration File

Logger settings defined in a program exist only in that program

Logger settings defined in a configuration file can be used by multiple programs
Sample Configuration File

# Use two loggers
handlers= java.util.logging.FileHandler, java.util.logging.ConsoleHandler

# Default global logging level.
.level= WARNING

# File logger default settings
# Default file output is in user's home directory (%h/).
# %g – use generation numbers to distinguish rotated logs
# limit = max size of each log file
# count = number of output files to cycle through
java.util.logging.FileHandler.pattern = %h/cs580Server%g.log
java.util.logging.FileHandler.limit = 50000
java.util.logging.FileHandler.count = 3
java.util.logging.FileHandler.formatter = java.util.logging.XMLFormatter

# Limit the message that are printed on the console to INFO and above.
java.util.logging.ConsoleHandler.level = INFO
java.util.logging.ConsoleHandler.formatter =
java.util.logging.SimpleFormatter

# Set levels of specific loggers
edu.sdsu.level = SEVERE
edu.sdsu.cs580.level = INFO
Using the Configuration File

Assume that configuration file is in
• Local directory
• In a file called cs580Log.properties

The following command will use the configuration file

```
java -Djava.util.logging.config.file=cs580Log.properties yourClassGoesHere
```
SDSU Logger

A number of Java logging systems exits for pre-JDK 1.4 Java

SDSU Java library contains one
SDSU Logger Example

import sdsu.logging.*;
public class LoggingExample
{
    public static void main(String[] args) throws LoggerCreationException {
        FileLogger.register( "MyLog");
        LoggingExample test = new LoggingExample();
        test.bar();
        test.foo();
    }

    public void foo() {
        Logger.log( "foo called");
    }

    public void bar(){
        try {
            Logger.log( "bar called");
            int a = 1;
            int b = 0;
            int c = a/b;
        }
        catch (Exception error) {
            Logger.error( "Error in bar");
            Logger.error( error);
        }
    }
}

MyLog.log contents

time='2:47:57 PM';date=10/2/00;type=Log;message='bar called';
time='2:47:57 PM';date=10/2/00;type=Error;message='Error in bar';
time='2:47:57 PM';date=10/2/00;type=Error;message='java.lang.ArithmeticException: / by zero';
time='2:47:57 PM';date=10/2/00;type=Log;message='foo called';
Types of Loggers

FileLogger - Sends log messages to file
NullLogger - Ignores log messages
ScreenLogger - Sends log messages to screen
SelectiveLogger - allows you to turn off types of messages

Types of Log messages

d debug
error
l log
w warning

Debug Class

Allows debug states to be turned off/on
Debug messages contain line number if JIT is off
Output can be sent to file or screen