Technical Manual
for Log Version 1.0.2

Klaus Hammermüller
klaus@ifs.tuwien.ac.at

October 7, 1999

Abstract

This paper is the technical documentation of the logging-mechanism used in the *Asgaard-framework* built on pure Java components. The focus is to concentrate distributed messages in a central log-file adding a complete instance of a Java-object to "freeze" it's state for further exploration or backup purpose.

Contents

1 Overview

1.1 What is Log?

Log is a collection of *pure Java*-classes implementing a logging mechanism for distributed applications and supports as well conventional logging as automated backup/restore-functionality (e.g. for the database) and debug-monitoring. *Log* can be implemented and used locally or access an remote running *Log*-server.

Features:

- Encapsulation the mechanism by only one class (called " `LogClient`");
- Transparent local or remote *Log*-server via RMI supporting a vary number of clients;
- The logfile is a human readable ASCII-flatfile and can be accessed as well via JDBC for full automated handling or processed by scripting languages;
- Scalable mailing mechanism to forward selected statements via e-mail;
- Automatically recovering of the logfile - the logfile can be removed at runtime;
- The log-file is human readable as well as completely automated recoverable. All logged parameters and objects can be restored;
- Some basic profiling capacities logging two time-stamps (local and remote) about the distributed environment.

1.2 Limitations

- Planned capacities as watchdog for client processes and restart capabilities have not been implemented yet;
- There is no support for different languages and no property files for storing default settings. The cause is to increase robustness of the components;
- Profiling capacities are weak in cause of the time consumed by the logging mechanism. The option driving the `LogClient` as thread seem to cost lot of performance, without this option the response-time depends on network load and disk access.

2 Install Log

2.1 Distribution

The Distribution is under GNU public license Version 2 `http://www.gnu.org/copyleft/gpl.html`, there is absolutely NO WARRANTY on this software.

A complete Distribution contains on the following parts:

- packages `asgaard.utils.log`, `asgaard.utils.encode` and `asgaard.utils.mail`;
- SimpleText flatfile JDBC driver;
- Suns JavaMail 1.1 classes.

2.1.1 Files

1. this document `Asgaard/doc/asgaard/utils/log`;
2. the binary distribution `bin/skid` (maintained with *Skid*);
3. guided by the installation `install/log` or this document;
4. the sources `Asgaard/src/asgaard/utils`;
5. and API documentation in `Asgaard/javadoc/asgaard/utils`.

2.1.2 SimpleText

This is an ASCII flatfile JDBC driver from `http://www.thoughtinc.com` which is used to generate the log-file.

```
Copyright:   (C) 1997 THOUGHT Inc.  All rights reserved.
Copyright:   (C) 1996 Karl Moss.  All rights reserved.
             You may study, use, modify and distribute this example
             for any purpose, provided that this copyright notice
             appears in all copies.  This example is provided WITHOUT
             WARRANTY either expressed or implied.
```

2.1.3 Mail

The mail-capabilities to inform automatically about failures insist the packages `mail.jar` and `activaion.jar` from SUN's free 1.1 Mail-Distribution which is available at `http://java.sun.com/products/javamail`.

2.2 Installation

Please follow the instructions on `http://install/log/index.html` or the following descriptions:

You must have at least JDK1.1.6 from JavaSoft or compatible Java environment at the Server you want to run Skid. In the rest of this page we assume that the Java interpreter (java) is installed correctly and available in your path.

2.2.1 *Log* installation

The following packages (Java-Bytecode) have to be copied to the server-machine and added to the CLASSPATH.

- The Log classes which are coming in `utils.jar` (which carries also other packages of the *Asgaard*-framework);
- The `SimpleText.zip` ASCII flatfile JDBC driver;
- the javamail 1.1 packages `activation.jar` and `mail.jar`;

3 Running *Log*

3.1 Command-Line Parameter

The remote *Log*-server `Log` is started with a Java virtual machine like

```
java asgaard.utils.log.Log
```

Command line parameter are [-hpLHMTm]:

```
-h              print this help message
-pL <LOGpath>   local directory to write the Logfile
-pH <mail HOST> used mailer (optional)
-pM <mail CC>   cc mailed statements (optional)
-pT <mail TO>   mailed statements to (optional)
-pK <LOGsize>   mail if the Logfile extends <LOGsize>kb
-m<statement>   type of statements which are forwarded
    0   forward no statements
    1   forward security statements
    2   forward error statements
    3   forward warning statements
    4   forward remark statements
    5   forward backup statements
    6   forward debug statements
```

The parameter are not stored, so need to be specified on every start. The mailer is only used if <mail HOST> and <mail TO> is known and <statement> ¿ 0. The <LOGpath> can be modified, the logfile itself has always the name log.sdf.

Before starting the Log-server the rmiregistry must be started to establish the *RMI* name-service.

3.2 Testing the Log-Installation

Sorry, there are no test-scripts built in with this distribution. To test the installation you have to use the *Log*-server with some implementation of the LogClient. (See next section.)

3.3 Starting the *Log*-server

A complete command-line call could look like:

```
rmiregistry
java -cp utils.jar;SimpleText.zip;activation.jar;mail.jar
     asgaard.skid.utils.Log
     -pL <path>
     -m3
```

If you are using the Logging mechanism locally in a program code no calls are necessary.

4 Integrate the LogClient into sources

The use of *Log* is encapsulated in one class named LogClient. For more detailed information please have a look at the javadoc-documentation.

4.1 Instantiation of a LogClient wrapping a remote *Log*-server

```
LogClient log = new LogClient("Some Prefix: ", "server-host", true);
```

Parameters:

1. "Some Prefix: " is used on System.err if any Exception occur as prefix;
2. "server-host" is the hostname of the remote *Log*-server;
3. asks if the client shall be started as thread.

4.2 Instantiation of a `LogClient` locally

```
LogClient log = new LogClient("Some Prefix: ", true);
```

The statement creates a local logfile with default settings without mailer.
Parameters:

1. `"Some Prefix: "` is used on `System.err` if any Exception occur as prefix;
2. asks if the client shall be started as thread.

4.2.1 Writing a *Log*-statement

```
log.put( "TaskName", "ServicePoint",
    LogMsg.LOG REMARK, true,
    "Message");
```

Parameters:

1. `"TaskName"` is the name of the process using the `LogClient`;
2. `"ServicePoint"` which was passed in the code creating the statement;
3. `"LogMsg.LOG REMARK"` type of log-statement;
4. asks if the servicepoint was passed successfully;
5. `"Messing"` adding some message;
6. Optional: adding some streamable object for backup.

4.3 The Logfile

The Logfile is a standard ASCII flatfile. First line starts with `.SDF` (the file-extension and following with the Fieldnames delimited by a semicolon " ,".

```
.SDF LOGNR,HOSTNAME,SERVERCLASS,SERVICEPOINT,TYPE,SUCCESS,LOGMESSAGE,CREATED,
7,'brunhild [128.130.167.58]','asgaard.utils.log.LogBroker','CreateLog','REMARK'
8,'nornen [128.130.167.57]','asgaard.skid.Skid','test_button','WARNING','ok'
```

This example is not complete. It The complete list of fields:

1. `LOGNR` (number) running number within the logserver;
2. `HOSTNAME` address of the host creating the statement;
3. `SERVERCLASS` Java class which created the statement;
4. `SERVICEPOINT` which was passed by creating the statement;
5. `TYPE` human readable type of the statement;
6. `SUCCESS` was the servicepoint passed sucessfully?
7. `LOGMESSAGE` a String
8. `CREATED` human readable time when the statement was created;
9. `TSDIFF` (number) difference between the two timestamps;
10. `RECEIVED` (number) time when the statement was processed;
11. `LOGTYPE` (number) type of the statement as number;
12. `BEANTYPE` (number) type of the bean (if any);
13. `LOGBEAN` encoded bytestream.

4.4 UML - Model of the Log-Components

A overview about all classes from package `asgaard.utils.log` is given by Figure 1. A short intro in the outlined classes:

- A *Log*-statement is carried by the `LogMsg` class which implements the *JavaBean* - conventions;
- The `LogBroker` is the (local) manager of *Log*-statements;
- `Log` wrappes the `LogBroker` as remote server;
- `LogClient` wrappes a local running `LogBroker` or a remote running `Log`-server by using the `RemoteLog`interface.

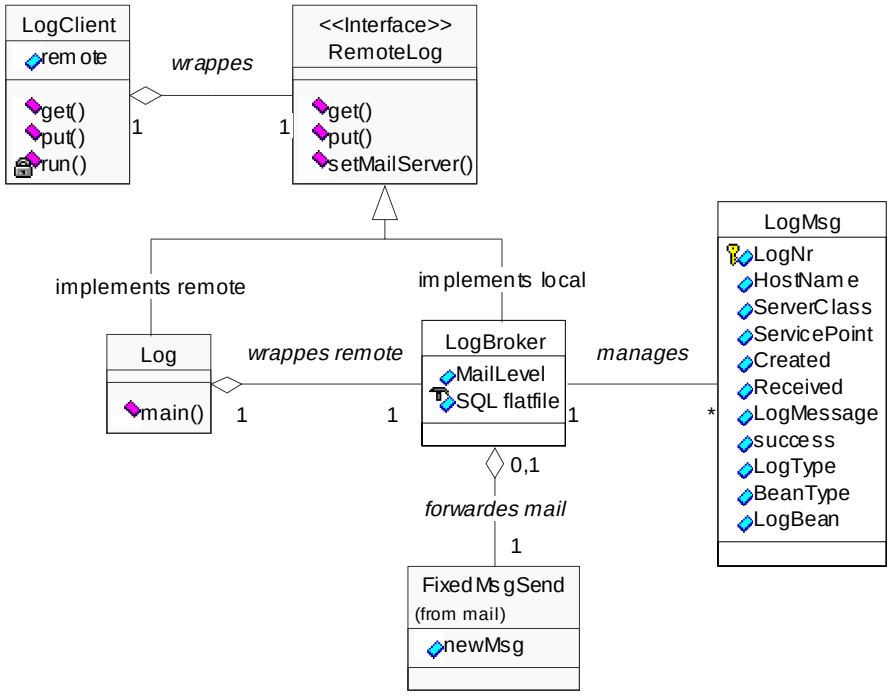

Figure 1: Class diagram from package `asgaard.utils.log`

To create the stub classes the remote `asgaard.utils.log.Log` class has to be compiled with the `rmic` compiler too.