KORA EVENT LOGGING SYSTEM USER'S MANUAL

Setting up the Configuration file (config.xml):

An example:

Note: XML is case sensitive and so is the configuration parser.

<config>

<setup algorithm="sha512" email_host='localhost' log_size="5"/>

The setup element is for configuring the overall system.
algorithm: the hash algorithm to use, “sha512”, “sha384”, “sha256”, “sha224”, “sha1”, and “md5” are supported. See http://docs.python.org/lib/module-hashlib.html for more details
email_host: the SMTP server to use for email notification upon job completion
log_size: the max size of an event log file in Kbytes.

<schedule run_job_every_x_days="40" job_start_time="2218" />

The schedule element is for determining when and how often the system runs.
run_job_every_x_days: run a fixity test with this many days in between. Must be at least 1 day.
job_start_time: time of day to start the fixity test in military time.
Note: the system will run this schedule starting today if **job_start_time** hasn't passed yet or will run as soon as job_start_time hits tomorrow**.**

<worker IP="35.9.22.153" path="Z:\" port="60000" />
<worker IP="35.9.22.158" path="/home/administrator/Desktop/matrix/" port="70000" />

Setup workers to execute a task.
The first example is a Windows machine at **IP** 35.9.22.153 listening on **port** 60000 with the local path of the archive available on Z:\.

The second example is of a Linux machine at **IP** 35.9.22.158 listening on **port** 70000 with /home/administrator/Desktop/matrix/ set as the local path to the archive, set to listen on port 70000. An arbitrary number of worker elements can be created.
Note: You MUST include the final slash (backslash on Linux and forward slash on Windows) for the **path** to be correct.

<database URL="localhost" dbname="matrix_pathdb" password="kora5" table="paths" username="root" filename="path" size="filesize"/>

The MySQL file path database in the archive that the system can access to retrieve filepaths. Most of these fields are self-explanatory but 'filename' is the attribute name in the designated table which gives the filename and path information to retrieve the file. 'size' is the name of the attribute with the file size in the table.

<notify email="chris_samiadjibenthin@yahoo.com" />
<notify email="bob@bob.com" />
</config>

The notify element email addresses of those who should be notified of fixity test completion. You can have an arbitrary number of addresses as they are stored in a list.

Basic Command Line Usage:

/usr/home/myname$ python Manager.py <options>

<options>

-a <filename>, add a file to the system

-r <filename>, remove a file from the system

-f , perform a single fixity test now

-s , run a scheduled fixity test

if no options are selected then the system will enter a menu with the following options:

0 => Run Fixity Test

1 => Run Schedule

2 => Add a File

3 => Remove a File

4 => Start XML-RPC server

otherwise => Quit

Options 0-3 perform the same functions as there command line counterparts, and the user will be prompted for a filename where appropriate.

Selecting Option 4, the user will be prompted for a port number then an XML-RPC server will be started which will listening to the selected port to receive remote commands. Note: the terminal window must be closed and re-opened for the port number to be released for use.

Main Components and What They Do:

-Manager.py: this is the main component of the Event Logging system it handles the distributed computing component of the checksum computations, updating a local sqlite database, and generating log events. Also it provides a simple command line user interface. This class will generate a new empty database if it gets erased.

-configParser.py: a component of the Manager, this class parses the configuration file (config.xml), puts it into dictionary form, and passes it to the Manager upon request. This will generate a default configuration file if it can't find it.

-LogGenerator.py: a component of the Manager as well, this class generates log files according to a user configurable size.

-ManagerServer.py: this class implements an XML-RPC server to support remote procedure calls. Each call is atomic (using mutex locks) to prevent corruption of the database by simultaneous read/write operations. The following API is supported:

fixityTest(), run perform a single fixity test

runSchedule(), run a scheduled job according to the configuration file. This starts on the day it's called, I.e. every 10 days at 1700 will start today at 1700 or tomorrow if it's already past 1700 today.

addFile(someFile), add somefile (a string) to the local database and checksum it.

removeFile(someFile), remove somefile (a string) from the local database.

forceLogDump(), force log files to be dumped out.

An Example XML-RPC client in Python

```
import xmlrpclib
server =  xmlrpclib.Server('http://localhost:8000)
server.addFile('foo.txt')
server.removeFile('foo.txt')
server.fixityTest()
server.runSchedule()
server.forceLogDump()
```

Software Requirements:

-Python 2.5 (www.python.com)

-there are 2 non-standard Python libraries used by this system:

a) parallel python (www.parallelpython.com), distributed computing framework

b) MySQLdb (sourceforge.net/projects/mysql-python), MySQL interface support

-an SMTP server needs to be available on the 'Manager' machine for email notification to be possible. We used Pythons' smtplib library to provide this functionality.

-it's suggested that the Manager be run on Linux but the Workers can be either Windows or Linux

Hardware:

-this software was tested on Pentium 4 machines with 1 GB RAM. Hard disk size should be large enough to store a database table of filename, checksum, and time stamp information for each file in the archive- generally this is a relatively trivial amount: 700 files needs around 270 KB in the database file.