Logging Framework Document

Oracle FLEXCUBE Private Banking

JULY 2013

ORACLE

Document Control

Author: Bhramara	Group: FCPB	
Created on : 29-MAY-2007	Revision No : 1	
Updated by : Mahendran Pandian	Reviewed by : VJ	Approved by:
Updated on : 02-JUL-2012	Reviewed on : 02-JUL-2013	Approved on : 02-JUL-2013

Table of Contents

1 Logging Framework

Concept

Logging constitutes inserting statements into the program that provide some kind of output information that is useful to the developer.
The logging framework for FCPB consists of:

- Jakarta Commons Logging (JCL): It provides a *Log* interface that is intended to be both light-weight and an independent abstraction of logging toolkits. It provides a middleware/tooling developer with a simple logging abstraction, which allows the application developer to plug in a specific logging implementation.

- Log4J: The underlying logging implementation.

Jakarta Commons Logging (JCL)

JCL provides thin-wrapper *Log* implementations for Log4J among other logging kits. There are two base abstractions used by JCL: *Log* (the basic logger) and *LogFactory* (which knows how to create Log instances).

- A particular *Log* implementation can be specified. But placing the commons-logging.jar in the classpath also results in JCL configuring itself in a reasonable manner.
- As for *LogFactory* implementation, the default implementation is sufficient which uses the following discovery process to determine what type of *Log* implementation it should use.
  - Use a factory configuration attribute named org.apache.commons.logging.Log to identify the requested implementation class.
  - Use the org.apache.commons.logging.Log system property to identify the requested implementation class.
  - If *Log4J* is available, return an instance of org.apache.commons.logging.impl.Log4JLogger **(which is the case with FCPB)**.
  - If *JDK 1.4 or later* is available, return an instance of org.apache.commons.logging.impl.Jdk14Logger.
  - Otherwise, return an instance of org.apache.commons.logging.impl.SimpleLog.

Since Log4j, the underlying implementation being used, is also the JCL primary default, any need for providing an implementation for either *Log* or *LogFactory* is eliminated.

Log4j

Log4j allows the developer to control which log statements are output with arbitrary granularity. Through proper configuration, it offers a hierarchical way to insert logging statements within a Java program. Multiple output formats and multiple levels of logging information are available. It is fully configurable at runtime using external configuration files. The configuration file used in FCPB for the same is log4j.xml. Log4j also maintains the log statements in the shipped code.

Log4j has three main components:

Loggers: Handle a majority of the logging operations.
Appenders: Control the output of the logging operations.
Layouts: Format the output for the appenders.

These three types of components work together to enable developers to log messages according to message type and level (**Loggers**), to control at runtime how these messages are formatted (**Layouts**) and where they are reported (**Appenders**).

1.1.1 Loggers

The most important advantage of using logging API is the flexibility available to disable or enable certain log statements, according to the choice of the developer. This is provided through **loggers** and **levels** associated with them. Loggers are named entities. The names of the same are case sensitive and they follow a hierarchical naming rule:

> ***"A logger is said to be an ancestor of another logger if its name followed by a dot is a prefix of the descendant logger name. A logger is said to be a parent of a child logger if there are no ancestors between itself and the descendant logger."***

For example, the logger named com.iflexsolutions.wm is the parent of the logger named com.iflexsolutions.wm.infra. The most important among the loggers is the root logger. It is mandatory to configure the same in the log4j.xml file, to be referred to, in case of no other loggers being defined for use.

E.g.: If a message is being logged from com.iflexsolutions.wm.refdata.core, and a logger is not defined in the log4j.xml, it looks for loggers for com.iflexsolutions.wm.refdata, com.iflexsolutions.wm, com.iflexsolutions in the same order. If none of these have been defined, the root logger is referred to.

log4j xml configuration for root logger:

```
<root>
  < level value="DEBUG" />
  <appender-ref ref="ConsoleAppender"/>
</root>
```

1.1.1.1 Levels associated with loggers:

There are 5 levels available for use with loggers.
DEBUG, INFO, WARN, ERROR, FATAL; DEBUG being of the lowest level and FATAL, the highest. All defined loggers will be associated with one of these levels as needed. In addition any logger will only output those messages which are of a level greater than or equal to it. If a level is not defined for a logger, it inherits the level of the closest ancestor whose level is set. In case of no ancestors found with a set level, it refers to the root logger for the same. The root logger is assigned the level, DEBUG, to enable logging all encountered messages.

log4j xml configuration for root logger, level.

```
<root>
  <level value="DEBUG" />
  <appender-ref ref="ConsoleAppender"/>
</root>
```

1.1.2 Appenders and Layouts

Log4j allows the logs to be printed to multiple destinations, each destination being referred to as **Appender**. In addition to destination selection, log4j also offers customization of the format in which the messages are logged. **Layout** provides the same. In effect, *layout* formats the messages being logged and *appender* sends the same to the defined destination(s).

Several Appenders available for use are:

- ConsoleAppender: appends log events to System.out or System.err using a layout specified by the user. The default target is System.out.
- FileAppender appends log events to a file.
- DailyRollingFileAppender extends FileAppender so that the underlying file is rolled over at a user chosen frequency.
- RollingFileAppender extends FileAppender to backup the log files when they reach a certain size.
- WriterAppender appends log events to a Writer or an OutputStream depending on the user's choice.
- SMTPAppender sends an e-mail when a specific logging event occurs, typically on errors or fatal errors.
- SocketAppender sends LoggingEvent objects to a remote a log server, usually a SocketNode.
- SocketHubAppender sends LoggingEvent objects to a set of remote log servers, usually a SocketNodes
- SyslogAppendersends messages to a remote syslog daemon.
- TelnetAppender is a log4j appender that specializes in writing to a read-only socket.

Layouts that are associated with Appenders are:

- HTMLLayout formats the output as a HTML table.
- PatternLayout formats the output based on a conversion pattern specified, or if none is specified, the default conversion pattern.
- SimpleLayout formats the output in a very simple manner; it prints the Level, then a dash '-' and then the log message.

1.1.3 Log4j.xml

The configuration file which configures the entire logging framework is the log4j.xml file.

1.1.3.1 Appenders and Layouts in FCPB

The Appenders and Layouts currently being used in FCPB are:
Appenders: ConsoleAppender, RollingFileAppender.
Layouts: PatternLayout.

log4j.xml configuration for ConsoleAppender:

```
<appender name=" ConsoleAppender" class="org.apache.log4j.ConsoleAppender">
  <layout class="org.apache.log4j.PatternLayout">
    <!-- Pattern in which the output is printed to the console-->
    <param name="ConversionPattern" value="%p - %C{1}.%M(%L) | %m%n"/>
  </layout>
</appender>
```

ConversionPattern: The pattern, in which the messages are logged, derived from the conversion characters defined in org.apache.log4j.PatternLayout.

For e.g.: **%**-5p means the priority of the logging event should be left justified to a width of five characters.

log4j.xml configuration for RollingFileAppender:

```
<appender name="INFRA_INFO" class="org.apache.log4j.RollingFileAppender">
    <param name="File" value="D:/logs/infra/infra_debug.log"/>
    <param name="Append" value="true"/>
    <param name="MaxFileSize" value="1MB"/>
    <param name="MaxBackupIndex" value="2"/>
    <layout class="org.apache.log4j.PatternLayout">
       <param name="ConversionPattern"
          value="%p - %C{1}.%M(%L) | %m%n"/>
    </layout>
    <filter class="org.apache.log4j.varia.LevelRangeFilter">
       <param name="LevelMin" value="DEBUG"/>
       <param name="LevelMax" value="INFO"/>
       <param name="AcceptOnMatch" value="true"/>
    </filter>
    <filter class="org.apache.log4j.varia.DenyAllFilter"/>
 </appender>
```

The param name-value of “File” is used to configure the location of the file into which the corresponding messages need to be logged.
The param name-value of “Append” is used to configure file truncation. The default value for this variable is true; meaning that by default a RollingFileAppender will append to an existing file and not truncate it.
The param name-value of “MaxFileSize” is used to configure the maximum size that the output file is allowed to reach before being rolled over to backup files. Default is 10MB.
The param name-value of “MaxBackupIndex” is to configure the number of back-up files to keep. Default is 1.
The configuration for “layout” is to define the pattern in which the messages will be logged.
The configuration for “filter” is to define the range of log levels to be logged into the corresponding appender. “LevelMax” and “LevelMin” define the range of log levels to log.

log4j.xml configuration to log into multiple destinations:

```
<logger name="com.iflexsolutions.wm.infra">
<level value ="DEBUG"/>
<appender-ref ref=" INFRA_INFO”/>
<appender-ref ref=" INFRA_WARN"/>
</category>
```

According to the above configuration any messages logged from com.iflexsolutions.wm.infra package will be sent to all 4 appenders. Further settings decide the specific logging.

Summarizing, each enabled logging request for a given logger, with the formatting, according to the defined Layouts, will be forwarded to all the Appenders in that logger as well as the Appenders higher in the hierarchy, unless the additivity flag is set to <u>false</u> overriding the default <u>true</u>.

Any java class in FCPB can use the available logging framework through the configured JCL SPI as shown in the below code snippet.

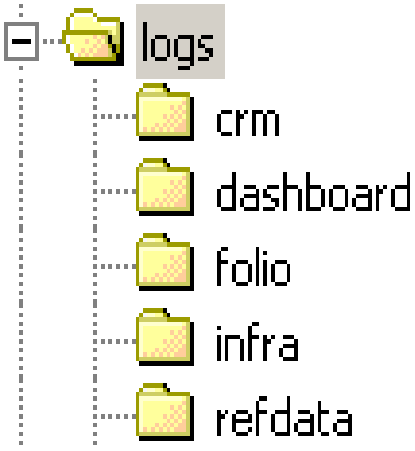
Code snippet to log messages

```
// Logging Imports
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class WMUniversalManagerImpl implements IWMUniversalManager {
    protected final Log log = LogFactory.getLog(getClass());
// provides a handle to the logFactory to log messages.
}

log.debug(java.lang.Object  obj);
log.warn(java.lang.Object  obj);
log.info(java.lang.Object  obj);
log.fatal(java.lang.Object  obj);
log.error(java.lang.Object  obj);
```

can be included in any class which extends WMUniversalManagerImpl for logging the messages.

Prerequisite

The folder structure



In the D drive of the local machine.

JULY 2013

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com/ financial_services

Copyright © 2012 Oracle Financial Services Software Limited. All rights reserved.

No part of this work may be reproduced, stored in a retrieval system, adopted or transmitted in any form or by any means, electronic, mechanical, photographic, graphic, optic recording or otherwise, translated in any language or computer language, without the prior written permission of Oracle Financial Services Software Limited.

Due care has been taken to make this Environment Details Document and accompanying software package as accurate as possible. However, Oracle Financial Services Software Limited makes no representation or warranties with respect to the contents hereof and shall not be responsible for any loss or damage caused to the user by the direct or indirect use of this Environment Details Document and the accompanying Software System. Furthermore, Oracle Financial Services Software Limited reserves the right to alter, modify or otherwise change in any manner the content hereof, without obligation of Oracle Financial Services Software Limited to notify any person of such revision or changes.

All company and product names are trademarks of the respective companies with which they are associated.