Enhydra Enterprise logging

Table of Contents

Chapter 1. Introduction

Enhydra Enterprise supports three different ways of logging. Logger interface defines abstract level of Enhydra logging (Logger and LogChannel interfaces)in eaf_api.jar (EAFApi module).

Old Standard logging system (to file and to console) is default and it is defined in application conf file by LogFile, LogToFile and LogToStderr parameters.

Also new Enhydra Enterprise supports log4j-logging. Implementations of Logger and LogChannel classes with log4j support are included in EAF.

Monolog is used in Jonas for logging. So, EAF contains Monolog implementation of Enhydra Logger. Implementations of Logger and LogChannel classes with log4j support are included in eafmonolog.jar.

If you want to use log4j or monolog logging, you have to set two new parameters in application configuration file : LogClassName and Log4j(Monolog).

LogClassName parameter means which implementation of logger class is used : Possible values:

- com.lutris.logging.StandardLogger (default value - for standard logging system)
- com.lutris.logging.Log4jLogger (for log4j logging)
- com.lutris.logging.MonologLogger (for monolog logging)

Log4j : Pathname of Log4j XML configuration file.

Monolog : Pathname of Monolog configuration file (trace.properties for Jonas J2EE Server).

Chapter 2. Enhydra logging - using log4j

Log4j allows logging requests to be printed to multiple destinations. In log4j speak, an output destination is called an appender.

By default Enhydra Application uses appenders to the console (if the Servlet container is instaled as console application) and to the log file (multiserver.log file in <ENHYDRA_ROOT>/multiserver/logs directory).

Chapter 3. Log4j configuration

This is very short explanation of log4j configuration rules. Detailed specification can be found at: http://jakarta.apache.org/log4j/docs/documentation.html
[http://jakarta.apache.org/log4j/docs/documentation.html].

Log4j has three main components: loggers, appenders and layouts. Loggers are named entities (com.foo is a parent of the logger named com.foo.Bar). The root logger can be configured with (log4j.xml):

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j='http://jakarta.apache.org/log4j/'>
    <appender name="ROLL" class="org.apache.log4j.RollingFileAppender">
        <param name="File" value="C:/enhydra-enterprise/multiserver/logs/enhydra.log"/>
        <param name="MaxFileSize" value="10MB"/>
        <param name="MaxBackupIndex" value="2"/>
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%d{ISO8601}: [%t] %C{1}, %p, %c: %m%n"/>
        </layout>
    </appender>
    <root>
        <level value ="info"/>
        <appender-ref ref="ROLL"/>
    </root>
</log4j:configuration>
```

where <level> is one of the: ALL, DEBUG, INFO, WARN, ERROR, FATAL, OFF.

ALL includes all messages, OFF does not show any message, DEBUG is for verbose output and with FATAL only critical errors should be logged.

Log4j also allows logging requests to print to multiple output destinations called appenders appender-(<ref> in example above). One of the distinctive features of log4j is the notion of inheritance in loggers. For example, the output of a log statement of logger C will go to all the appenders in C and its ancestors. However, if an ancestor of logger C, say P, has the additivity flag set to false, then C's output will be directed to all the appenders in C and it's ancestors upto and including P but not the appenders in any of the ancestors of P. Loggers have their additivity flag set to true by default.

The target of the log output can be a file, an OutputStream, a java.io.Writer, a remote log4j server, a remote Unix Syslog daemon or even a NT Event logger among many other output targets.

Hence, appender type is mostly one of: ConsoleAppender, FileAppender, RollingFileAppender, Daily-RollingFileAppender, SocketAppender, JMSAppender, SMTPAppender, AsyncAppender.

If you want to customize not only the output destination but also the output format, it can be accomplished by associating a layout with an appender. For example, the PatternLayout with the conversion pattern %r [%t]%-5p %c - %m%n will output something akin to:

```
176 [main] INFO  org.foo.Bar - Located nearest gas station.
```

In the output above: the first field equals the number of milliseconds elapsed since the start of the program, the second field indicates the thread making the log request, the third field represents the priority of the log statement, the fourth field equals the name of the logger associated with the log request, the text after the - indicates the statement's message.

Chapter 4. Log4j configuration - example

For output logging requests printed to Windows NT Event Loggers (for Windows NT, Windows 2000 or Windows XP, versions of Windows which support Event Viewer program).

```
----------------------------------------------------------------------
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j='http://jakarta.apache.org/log4j/'>
    <appender name="ROLL" class="org.apache.log4j.RollingFileAppender">
        <param name="File" value="C:/enhydra-enterprise/multiserver/logs/enhydra.log"/>
        <param name="MaxFileSize" value="10MB"/>
        <param name="MaxBackupIndex" value="2"/>
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%d{ISO8601}: [%t] %C{1}, %p, %c: %m%n"/>
        </layout>
    </appender>
    <appender name="ACCESS" class="org.apache.log4j.RollingFileAppender">
        <param name="File" value="C:/enhydra-enterprise/multiserver/logs/acces.log"/>
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%m%n"/>
        </layout>
    </appender>
    <appender name="Console" class="org.apache.log4j.ConsoleAppender">
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%d{ISO8601}: %p: %m%n"/>
        </layout>
    </appender>
    <appender name="EventViewer" class="org.apache.log4j.nt.NTEventLogAppender">
        <param name="source" value="Enhydra"/>
        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%d{ISO8601}: [%t] %C{1}, %p, %c: %m%n"/>
        </layout>
    </appender>
    </appender>
    <logger name="REQUEST">
        <appender-ref ref="ACCESS"/>
    </logger>
    <logger name="SysOut">
        <appender-ref ref="Console"/>
    </logger>
    <root>
        <level value ="info"/>
        <appender-ref ref="ROLL"/>
        <appender-ref ref="Console"/>
        <appender-ref ref="EventViewer"/>
<!--
        <appender-ref ref="XMLOutFormat"/>
        <appender-ref ref="CHAINSAW_CLIENT"/>
-->
   </root>
</log4j:configuration>
----------------------------------------------------------------------
```

- When the code is changed, save log4j.xml to <ENHYDRA_ROOT> and restart Enhydra Enterprise again.
- Then, start Windows Event Viewer. Go to the Application's Log. All output messages (from Enhydra Enterprise Server) will be shown in the window.

If you want to change 'source' parameter in Event Viewer (application name instead of 'Enhydra'), then you should create separate logger for each application you want to log with it's own 'source' name. The name of the logger should be equal to Server.AppClass value in conf file of that application, i.e. the last name of names separeted by dots with inserted 'org.enhydra'. Logger and its appender definitions are given in the following example. If application's conf file (golfshop.conf) contains line:

```
Server.AppClass=golfshop.GolfShop
```

then log4j.xml should contain following lines among <appender> elements:

```
<appender name="EventViewerGolfShop" class="org.apache.log4j.nt.NTEventLogAppender">
    <param name="source" value="Enhydra-GolfShop"/>
    <layout class="org.apache.log4j.PatternLayout">
        <param name="ConversionPattern" value="%d{ISO8601}: [%t] %C{1}, %p, %c: %m%n"/>
    </layout>
</appender>
```

and following lines among <logger> elements:

```
<logger name="org.enhydra.GolfShop" additivity="false">
    <appender-ref ref="EventViewerGolfShop"/>
</logger>
```

Chapter 5. Enhydra logging - using Monolog

Monolog allows logging requests to be printed to multiple destinations. Monolog contains several implementations for Monolog interface. As a default (in Jonas and in Enhydra Enterprise), log4j implementation of Monolog logging is used.

Monolog Logging can be configured as following:

- User configuration

  A user can define a Monolog key in the application configuration file. This key should point to the logger configuration file. As well, a LogClassName property should be defined (com.lutris.logging.MonologLogger). An example (AirSent.conf):

  ```
  LogClassName = com.lutris.logging.MonologLogger
  Monolog = C:/trace.properties
  ```

- Default

  If no user configuration is done, logger initialization searches the classpath for a valid logger configuration file. The name of the the configuration file can be changed in JOnAS property file. This file must exist in the classpath. A configuration file name example (jonas.properties):

  ```
  jonas.log.configfile trace
  ```

The configuration of monolog is described by several properties. It is therefore possible to define:

- Levels: you can define intermediate levels which you can use to initialize logger level.
- Handlers: An handler represents an output. Monolog provides four standard handlers(console, file, RollingFile, and ntevent) and a generic handler which permits to configure any handler.
- Loggers: A Logger is identified by names. However we consider that each logger has a main name. This name is used to identify it in the property file. There are several configurable things on a Logger instance.

To each logged message, a level must be associated. This level characterises the importance of the level. The monolog specification defines six levels:

- FATAL: It characterizes a very high error message.
- ERROR: It characterizes an error message.
- WARN: It characterizes a warning message.
- INFO: It characterizes a information message.

- DEBUG: It characterizes a debugging message.
- INHERIT: This level does not characterize the importance of a message but it is used to configure a logger. It permits to specify that a logger must inherit its level of its ancestor.

More details about Monolog logging can be found at: http://monolog.objectweb.org/index.html [http://monolog.objectweb.org/index.html].

Chapter 6. Tomcat logging - using Monolog

Application Logging

In orded to perform logging in Tomcat, Enhydra uses its extension of Jakarta Commons Logging.

There are two ways of configuring Monolog logging in Tomcat.

- Default configuration

  By default Monolog logging in Tomcat is initialized by a search for a property file in the classpath. The name of the property file is defined in the JOnAS under a following property key:

  ```
  jonas.log.configfile
  ```

  Here it is possible to change the name of the file. In that case it must exist in the classpath. All the neccesary configuration of the logger can be done in this property file.

- User configuration

  A user can define a Monolog key in the application configuration file. This key should point to the logger configuration file. As well, a LogClassName property should be defined (com.lutris.logging.MonologLogger). An example:

  ```
  LogClassName = com.lutris.logging.MonologLogger
  Monolog = C:/trace.properties
  ```

Server Logging

MonologFileLogger

MonologFileLogger is an extension of org.apache.catalina.logger.LoggerBase. It is used instead of the org.apache.catalina.logger.FileLogger to stream the Tomcat's logging trough Monolog. The configuration of the logger is straight forward. In the *server.xml* a parameter should be declared:

```
      <Logger className="org.apache.catalina.logger.MonologFileLogger" />
```

The rest of the configuration is done in the trace.properties configuration file. If no specific logger for MonologFileLogger is declared, the setting for the root logger will be used.

Chapter 7. Jetty logging - using Monolog

Monolog logging in Jetty is performed using Monolog's extension of Jetty's LogSink class: MonologSink.

To enable Monolog logging in Jetty, the jetty.xml file in the enhydra/conf folder should contain the following lines:

```
<Call name="instance" class="org.mortbay.util.Log">
    <Call name="disableLog"/>
    <Call name="add">
      <Arg>
        <New class="org.enhydra.logging.MonologSink">
          <Call name="start"/>
        </New>
      </Arg>
    </Call>
  </Call>
```

- Default configuration

  By default Monolog logging in Jetty is initialized using the property file found in the classpath. Therefore, all the necessary configuration should be done in this file. The name of the file depends on the settings in JOnAS - jonas.properties and the key jonas.log.configfile and can be therefore changed as needed. An example:

  ```
  jonas.log.configfile trace
  ```

  The only restriction is that this file must be present in the classpath.