DeviceInterface Logging Architecture

Version: 1.0 | 01/17/2013

Copyright 2012 Barco Inc.

All rights reserved.

This document is protected by copyright and distributed under licenses restricting its use, copying, distribution and decompilation. No part of this document may be reproduced in any form by any means without prior written authorization of Barco.

Contents

[Document History 2](#_Toc346811853)

[Introduction 3](#_Toc346811854)

[Logging Architecture 4](#_Toc346811855)

[Event Levels 4](#_Toc346811856)

[Configuration 5](#_Toc346811857)

[Appender 5](#_Toc346811858)

[Logger Category 6](#_Toc346811859)

[PatternLayout 6](#_Toc346811860)

[Log4JErrorHandler 6](#_Toc346811861)

[Adding your own Logger 6](#_Toc346811862)

[Logging - API 7](#_Toc346811863)

[DeviceFrameworkI 7](#_Toc346811864)

[MessageBusListenerI 7](#_Toc346811865)

[Constants 7](#_Toc346811866)

# Document History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Reason for change |
| 1.0 | 01/17/2013 | CGAN | internal release |

# Introduction

This document provides a high level overview of the Logging Architecture used in DeviceInterface SDK. It proposes the use of Log4j as the logging API.

# Logging Architecture

Interface com.barco.device.common.DeviceFrameworkI provides the following methods for logging.

public interface DeviceFrameworkI extends MessageBusListenerI{

. . . .

. . . .

/\*\*

\* Logs the message trace

\*/

public abstract void messageTrace(String logMessage, Message message, Throwable ex);

/\*\*

\* Returns a log4j Logger instance

\*/

public abstract Logger getLogger();

. . . .

}

getLogger returns a log4j Logger instance which is used to log at the desired level (*see Event Levels*). The ‘messageTrace’ method allows to log the logMessage string at the level specified in the Message object.

This allows us to log at different level on a per message basis if need be.

The class that implements these log methods is DeviceImpl. There are three types of logger categories defined in the default log4j configuration file that comes with the deviceinterface module:

* CONSOLE – logs to system.out
* DAEMON– logs to sysout.
* CONSOLE\_DAEMON – logs to both.

By default the logger category defined is CONSOLE\_DAEMON.

This can be changed as per code requirements using ‘setLoggerCategory(String category)’ method defined in the deviceImpl class.

public void setLoggerCategory(String category){. . . .}

## Event Levels

* ALL: This is the lowest possible level associated with log4j and is intended to turn on all logging.
* TRACE: Intended for fine grained tracing information such as entry and exit from a method with parameters.
* DEBUG: Intended for logging variable values to help the developers.
* INFO: Intended to log informational messages to highlight the progress of the application.
* WARN: Designates the potentially harmful situations in the application.
* ERROR: Errors in the application that might still allow the application to continue running.
* FATAL: Designates very severe error events that will presumably lead to application abort.

Log4j assumes that levels are ordered. For the standard levels, we have TRACE < DEBUG < INFO < WARN < ERROR < FATAL.

## Configuration

Log4j supports different ways to configure the logging in the system:

* Through a configuration file written in XML
* Through a configuration property file

Out of these, the supported method for configuration is through an XML file defined outside the application classpath. Log4j will load this configuration by means of the java system property that is defined while executing the DeviceInterface jar:

‘java -Dlog4j.configuration=path/to/log4j.xml’

Log4j will inspect the "log4j.configuration" system property and, if set, will attempt to load the configuration. If no configuration file could be located the DefaultConfiguration will be used. This will cause logging output to go to the console with default level of ERROR.

<?xml version=*"1.0"* encoding=*"UTF-8"* ?>

<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration xmlns:log4j=*"http://jakarta.apache.org/log4j/"*>

<appender name=*"console"* class=*"org.apache.log4j.ConsoleAppender"*>

<errorHandler class=*"com.barco.logger.Log4JErrorHandler"*/>

<param name=*"Target"* value=*"System.out"* />

<layout class=*"org.apache.log4j.PatternLayout"*>

<param name=*"ConversionPattern"* value=*"%d %-5p - %m%n"* />

</layout>

</appender>

<category name=*"CONSOLE"*>

<priority value=*"debug"* />

<appender-ref ref=*"console"* />

</category>

<root>

<priority value=*"error"* />

</root>

</log4j:configuration>

This is a sample configuration which demonstrates the use of appenders, loggers and layout being used in the log4j configuration. This is a snippet from the configuration file provided with DeviceInterface.

### Appender

There are several appenders defined in log4j. The sample shows a ConsoleAppender implemented by the class ‘org.apache.log4j.ConsoleAppender’ in log4j. The properties for the appenders are elements defined inside the appender tag.

You can also define other types of appenders for log4j. Example of a SysLogAppender is:

<appender name=*"syslog"* class=*"org.apache.log4j.net.SyslogAppender"*>

<errorHandler class=*"com.barco.logger.Log4JErrorHandler"*/>

<param name=*"Facility"* value=*"LOCAL7"*/>

<param name=*"FacilityPrinting"* value=*"true"*/>

<param name=*"SyslogHost"* value=*"localhost"*/>

<layout class=*"org.apache.log4j.PatternLayout"*>

<param name=*"ConversionPattern"* value=*"%d %-5p - %m%n"* />

</layout>

</appender>

### Logger Category

There is a way to define different categories of loggers for different appenders. The sample shows a category for Console appender:

<category name=*"DAEMON"*>

<priority value=*"debug"* />

<appender-ref ref=*"syslog"* />

</category>

The priority value here designates the minimum level of log events that can be processed by this category. A level higher than the priority will be accepted and allowed to be passed to the appender reference for layout formatting and actual logging.

### PatternLayout

This defines the layout format for the log message handled by the appender. The sample layout shown above will create a log like:

‘2013-01-18 16:33:53,011 DEBUG - This is a log message!’

### Log4JErrorHandler

The Log4jErrorHandler class implements the log4j’s default error handling policy which consists of emitting a message for the first error in an appender and ignoring all following errors. This policy aims at protecting an otherwise working application from being flooded with error messages when logging fails.

### Adding your own Logger

Apart from the logger categories provided with the default Configuration file, you can choose to add your own by modifying the Configuration file.

The deviceImpl class assumes that there are atleast three basic logger categories defined which are CONSOLE, DAEMON and CONSOLE\_DAEMON.

You can choose to modify their priority levels and targets. If you need to add your own logger, add the same using the format described in the configuration section above. Now tell log4j to use this modified configuration file using system property ‘java -Dlog4j.configuration=path/to/log4j.xml’

## Logging - API

### DeviceFrameworkI

#### GetLogger

Parameters: NA

Return: Log4j Logger instance

#### SetLoggerCategory

Parameters: String logger type (CONSOLE/DAEMON/CONSOLE\_DAEMON)

Returns:NA

If you need a logger instance with Daemon logger, you can pass a type argument to the the setLoggerCategory method:

‘logger.setLogger(DeviceImpl.DAEMON)’

#### MessageTrace

Parameters: String log message, Message message, Throwable e

Returns:NA

If you want to log a message, you need to pass in the message string to log and the message object. Throwable can be null;

‘logger.messageTrace(logMessage, message, e)’

### MessageBusListenerI

#### HandleLog

Parameters: String log message, Level log level, Throwable e

Returns:NA

Allows the agent implementors to use this method to log at the desired level.

‘listener.handleLog(logMessage, Level.ERROR, e)’

### Constants

Logging levels are mapped from requests to log4j Levels.

DEBUG = 5;

WARN = 4;

INFO = 3;

ERROR = 2;

FATAL = 1;