org.jax.mgi.shr.log Design Document

Author: Mike Walker

Created: March 20, 2003

Last Modified: July 27, 2004 15:58

1 Purpose of Document

This document describes the classes belonging to the package org.jax.mgi.shr.log from the lib_java_core product and provides source code examples for common usage patterns.

2 Introduction

The purpose of the org.jax.mgi.shr.log package is to provide a logger interface and a logger factory interface for use in creating application loggers that can be plugged into frameworks classes such as the **SQLDataManager** and the **BCPManager** classes from the org.jax.mgi.shr.dbutils package. Additionally, a simple implementation of the **Logger** interface is provided which logs messages to the console through standard out.

3 Overview of Classes

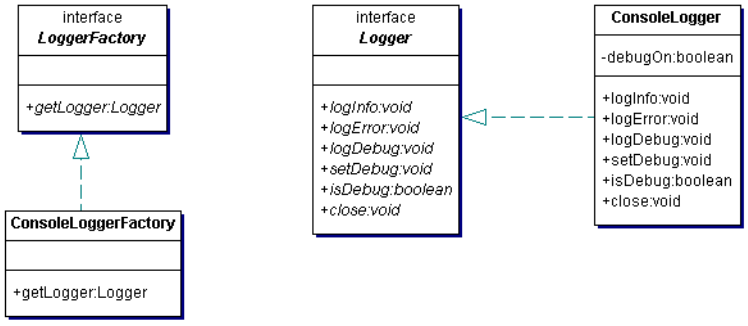
The **Logger** interface provides a simple logging interface that allows logging messages classified as one of several severity levels. These include debug, info and error. Additionally, the interface provides a method for turning debug messages off or on.

The **LoggerFactory** interface has one method, the getLogger() method. This interface is intended to be implemented by any developer who implements the **Logger** interface and it's intent is to give a implementation independent way of creating an instance of a **Logger**. The **LoggerFactory** class can then be plugged into the frameworks classes through the configuration. The **LogCfg** class is used by the frameworks classes to access the factory. The configuration variable is LOG_FACTORY.

The **ConsoleLogger** is an implementation of the **Logger** interface which routes messages to the application console.

The **ConsoleLoggerFactory** class creates a **ConsoleLogger** and is used as the default **LoggerFactory** class within the fameworks classes.

4 Class Diagram



5 Logger Interface

The logging methods provided by the **Logger** interface are logDebug(String), logInfo(String) and logError(String). Additionally, there are utility methods close(), isDebug(), and setDebug(boolean). The logging methods provide three levels of message severities, debug, info and error. It will be up to the implementation what to do with these severity levels. An example implementation is the **DLALogger** from the org.jax.mgi.shr.dla package. In this implementation, log messages are timestamped with the date and time, class and method name from which the message came, and severity level of the message.

The utility methods include a close() method. This is provided in case your implementation requires special resource closing actions. The setDebug(boolean) is provided as a way to turn debug messaging on or off. This way programmers can leave debug messaging in their code and decide at runtime whether or not they should be logged. The **SQLDataManager** from the org.jax.mgi.shr.dbutils uses the **Logger** interface. It prints all sql statements and their execution times to the log as debug messages. By default the debug messages are turned off. In order to turn them on, the user can set the LOG_DEBUG configuration parameter to true. The method isDebug() is provided as a way to test if the **Logger** is in debug state or not. It will return a boolean to indicate this.

6 LoggerFactory

This interface is used as an implementation independent way of creating a particular implementation of the **Logger** interface. It provides the single getLogger() method. It is used by most frameworks classes from the lib_java_core product for creating loggers. Besides having the setLogger(Logger) method, these frameworks classes also allow configuring your system with a designated **LoggerFactory** class. The frameworks classes will be able to instantiate the configured factory class and obtain a specific logger through the call to the getLogger() method. The configuration parameter used to set the **LoggerFactory** is LOG_FACTORY.

7 ConsoleLogger and ConsoleLoggerFactory

These classes are reference implementations of the **Logger** interface and the **LoggerFactory** interface. The **ConsoleLogger** writes messages to the application console. It does not differentiate between info level and error level. Both of these are simplified echoed to the console. The debug messages can be turned on or off by the setDebug(boolean) method. The frameworks classes within the lib_java_core product uses the **ConsoleLoggerFactory** by default.