Integrating 3rd Party Logging Frameworks into SAP NetWeaver

SAP Platform Ecosystem

Copyright

Icons in Body Text

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax

Additional icons are used in SAP Library documentation to help you identify different types of information at a glance. For more information, see *Help on Help → General Information Classes and Information Classes for Business Information Warehouse* on the first page of any version of *SAP Library*.

Typographic Conventions

Type Style	Description
Example text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options.
	Cross-references to other documentation.
Example text	Emphasized words or phrases in body text, graphic titles, and table titles.
EXAMPLE TEXT	Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.
`Example text`	Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example text`	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
`EXAMPLE TEXT`	Keys on the keyboard, for example, `F2` or `ENTER`.

Table of Contents

Integrating 3rd Party Logging into SAP Logging

Introduction

Logging is an important instrument for developers, administrators and the support of applications. SAP has a common Logging API which is used by the J2EE engine and all SAP Java applications, so all the logs have a common format and can be displayed and analyzed in the same viewer. For your applications it is recommended to make use of the SAP Logging API.

Nevertheless often you have to deal with existing applications or 3rd Party libraries that already make heavy use of a different Logging API. For instance Log4j is a good example for a wide spread logging API, which is used by many open source projects like JBoss, TheServerSide, Velocity and many more. So frequently you are in the uncomfortable situation where you have two logging solutions in your applications. The consequence of this coexistence brings at least a loss of quality for manageability, since there are two different concepts for logging with different destinations for the logging output.

So what to do, when you have to deal with components that use a different logging API? The ideal solution would be replacing the 3rd Party logging with SAP logging, which in practice not seldom means that you would have to change thousands lines of code. With that in mind, we've decided to find more practicable solutions for this common problem. The intention of this document is to provide easy solutions to integrate 3rd Party logging solutions into SAP logging, that allow you to benefit of the SAP logging without having to change your code. This goal is basically achieved by routing the log messages of the 3rd party loggers to SAP logging. This tutorial will show you exemplarily how this can be done for Log4j and Commons Logging. Beside that the tutorial enable you to write your own bridge to a logging API or modify the bridges of the tutorial, since it teaches you how this can be done.

Prerequisites

Systems, installed applications, and authorizations

- ☐ SAP Web AS – Java 6.40

Knowledge

- ☐ Understanding of SAP logging

Pitfalls

Although most logging APIs share basic concepts, there can be essential differences between the APIs. Therefore routing one's API messages to the SAP logging API forces you to make compromises. To get a feeling for the limitations and to decide if these limitations are acceptable in your context, an understanding of the concept of SAP logging is needed. Explaining SAP logging in detail is beyond the scope of this document. To make yourself familiar with SAP logging the Logging Chapter in the online help (http://help.sap.com) is a good starting point.

The following points list problems that you have to face, when you route messages of a 3rd Party Logging framework to SAP Logging. It is very important to be aware of these restrictions, since they influence the way you have to read your logs.

SAP logging draws a straight line between logs and traces. Logs are for administrators and traces for developers. Log-files have the ending "log" and trace-files "trc". Reproduction of this separation into SAP logging from logging solutions that do not have this differentiation can be tricky. You have to be aware of the way the bridge routes the messages, to know where to look for the entries.

Severities of the 3rd party logging API have to be mapped to SAP logging, which may lead to unwished results.

Timestamps are set at the time the bridge calls SAP logging instead of the time where the message was written by the 3rd party logging API.

Bridges of 3rd Party Logging to SAP Logging

In the next sections we will explain by example how a bridge from a 3rd party logging API to SAP logging can be realized. For the demonstration we've chosen Log4j and Commons Logging. As already mentioned, Log4j is wide-spread. Commons Logging is not a classical logging toolkit. It provides you an API for logging which abstracts the underlying logging solution, to get independent of a certain logging solution. The underlying logging solution can be easily changed by configuration without having to get hands on the source code. Caused by this added value, commons logging has wide acceptance in the open source community. For instance the very popular Apache Struts Web Application Framework uses Commons Logging. For these reasons we've decided to go for these frameworks. Nevertheless by reading this document and doing the examples you get a feeling what it takes to write a bridge to SAP logging, so for instance writing a handler for JDK 1.4 logging won't be hard.

Apache Log4j bridge

The Log4j framework writes the log messages to so called appenders. In a configuration file can be defined which appenders should be used. Out of the box Log4j comes with a few appenders like Console Appender, File Appender and many more. For our tutorial we've written a new appender that writes the messages to SAP logging and configured Log4j to use our appender in the *log4j.properties* file which you can see below.

Log4j.properties

```
# Configuration of: 1) rootLogger Log-Level 2) Appenders
log4j.rootLogger=debug, SAPLogging

#=== Configuration SAP Logging Appender ===
# The appender class
log4j.appender.SAPLogging=com.sap.logging.bridge.log4j.SapLogAppender
# Name of the SAP Logging Category under "Applications"
log4j.appender.SAPLogging.categoryName=MyCategory
```

The file tells log4j, that the root logger should write out all messages that have at least the debug-level. The only appender (destination for your log messages) is our SAPLogging-Appender, which can be loaded by Log4j by using the fully qualified name of our appender class.

In the last line of the file is a parameter *categoryName*, that defines the name of the Category under the Application node to which the log messages will be written, which is important for you to know when you are searching for the log messages in the LogViewer.

The following table shows how the severities are mapped and where the messages are written to.

Log4j	SAP logging
Debug	Debug written to Trace
Info	Info written to Log (Category defined in Log4j-Configuration file)
Warn	Warning written to Log (Category defined in Log4j-Configuration file)
Error	Error written to Log (Category defined in Log4j-Configuration file)
Fatal	Fatal written to Log (Category defined in Log4j-Configuration file)
Other (i.e. user defined levels)	Warning written to Log (Category defined in Log4j-Configuration file)

All debug messages will go to SAP Traces and everything else will go to the configured Category log. Unknown levels like user defined levels or the new *Trace* level of Log4j version 1.2.12 will get the Severity *Warning*. If you already use Log4j version 1.2.12 we recommend to map the Trace level to SAP Severity Debug. In the demo the Trace level isn't handled, because we expect you use an older Log4j version.

For a deeper understanding take a look at the Appender Source code where you can quickly find the according sections to the table.

Jakarta Commons Logging

The basic idea of Commons Logging is to offer an API for logging that is implemented by the different logging toolkits, so the logging toolkits can be changed without touch the source codes. So if you want to use SAP logging with Commons Logging you need to have an implementation class, that we've created for this tutorial (in the web project in the example: *com.sap.logging.bridge.jcl.SapLogJclImpl*).

Which Implementation is used can be set in a configuration file called *commons-logging.properties* that has to be in the classpath during runtime.

commons-logging.properties

```
org.apache.commons.logging.Log=com.sap.logging.bridge.jcl.SapLogJclImpl
```

As you can see the name of the category to which log entries are written is not configured here. For the sake of simplicity and concentration on the essence the category name is hard coded in the Implementation class. Feel free to implement a reading out of a configuration file.

Commons Logging basically asks the implementer to implement the methods listed in the following table, that shows how these methods are mapped to SAP logging.

JCL-Method	SAP logging – Severity and destination for message
trace	Debug written to Trace
debug	Debug written to Trace
info	Info written to Log (Category defined in implementation class)
warn	Warning written to Log (Category defined in implementation class)
error	Error written to Log (Category defined in implementation class)
fatal	Fatal written to Log (Category defined in implementation class)

As you can see the concept for routing the messages is basically the same for Log4j and Commons Logging.

The sample application

This tutorial comes with two NetWeaver Developer Studio Projects. Where to get these projects and how to work with them is discussed in the following chapter *Running the Sample*. This chapter gives you an overview of the projects.

There is an Enterprise Application Project *TutLogging_Integration_Ear* and a Web Module Project *TutLogging_Integration_Web.* The Enterprise Application Project's purpose is to package the Web Module into a deployable EAR-file.

The web module basically consists of two JSPs, a few Java classes and two properties configuration files. The JSP pages are for triggering some logging. By loading these pages you simply call a method in a Java class that either calls Log4j logging API or Commons Logging API depending on the JSP-Site (see following source codes *`Log4jExampleUsage`* and *`JclExampleUsage`*). Log4j and Commons logging are configured in properties files to write the messages to the SAP logging API.

The following source codes show you how the 3rd Party Logging is used.

Class Log4jExampleUsage

```
package org.example.app;
import org.apache.log4j.Logger;
/**
 * Does some logging with log4j API for demonstration.
 */
public class Log4jExampleUsage {

   /** The log4j-Logger*/
   private static final Logger log4j =
Logger.getLogger("org.example.app.Log4jExampleUsage");
   /**
    * Just some log4j log output.
    */
   public static void logSomething() {
      // traces
      log4j.debug("Log4j - debug-method");
      log4j.debug(
         "Log4j - debug-method with exception",
         new NullPointerException("Developer's best friend."));
      //logs
      log4j.info("Log4j - info-method");
      log4j.warn("Log4j - warn-method");
      log4j.error("Log4j - error-method");
      log4j.fatal("Log4j - fatal-method");
      log4j.fatal(
         "Log4j - debug-method with exception",
         new NullPointerException("Developer's best friend."));
   }
}
```

Class JclExampleUsage

```java
package org.example.app;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

/**
 * Does some logging with commons logging API for demonstration.
 */
public class JclExampleUsage {

   /** The JCL-Logger*/
   private static Log jcl = LogFactory.getLog(JclExampleUsage.class);


   /**
    * Just some commons logging log output.
    */
   public static void logSomething() {
      // traces
      jcl.debug("JCL - debug-method");
      jcl.debug(
         "JCL - debug-method with exception",
         new NullPointerException("Developer's best friend."));

      //logs
      jcl.info("JCL - info-method");
      jcl.warn("JCL - warn-method");
      jcl.error("JCL - error-method");
      jcl.fatal("JCL - fatal-method");
      jcl.fatal(
         "JCL - debug-method with exception",
         new NullPointerException("Developer's best friend."));
   }
}
```

So after you've triggered the methods in the the classes above you'll have the following messages in your Log Viewer.

Time	Category	Date	Message
11:28:04:454	/Applications/M...	10/06/2005	JCL - debug-method with exception [EXCEPTION] java.lang.NullPointerException: Develop
11:28:04:454	/Applications/M...	10/06/2005	JCL - fatal-method
11:28:04:454	/Applications/M...	10/06/2005	JCL - error-method
11:28:04:454	/Applications/M...	10/06/2005	JCL - warn-method
11:28:04:444	/Applications/M...	10/06/2005	JCL - info-method
11:28:04:444		10/06/2005	JCL - debug-method with exception [EXCEPTION] java.lang.NullPointerException: Develop
11:28:04:444		10/06/2005	JCL - debug-method
11:27:46:038	/Applications/M...	10/06/2005	Log4j - debug-method with exception [EXCEPTION] java.lang.NullPointerException: Develc
11:27:46:038	/Applications/M...	10/06/2005	Log4j - fatal-method
11:27:46:038	/Applications/M...	10/06/2005	Log4j - error-method
11:27:46:038	/Applications/M...	10/06/2005	Log4j - warn-method
11:27:46:038	/Applications/M...	10/06/2005	Log4j - info-method
11:27:46:028		10/06/2005	Log4j - debug-method with exception [EXCEPTION] java.lang.NullPointerException: Develc
11:27:46:008		10/06/2005	Log4j - debug-method

Picture 1 defaultTrace.trc

Time	Category	Date	Message
11:28:04:454	/Applications/MyCategory...	10/06/2005	JCL - debug-method with exception
11:28:04:454	/Applications/MyCategory...	10/06/2005	JCL - fatal-method
11:28:04:454	/Applications/MyCategory...	10/06/2005	JCL - error-method
11:28:04:454	/Applications/MyCategory...	10/06/2005	JCL - warn-method
11:28:04:444	/Applications/MyCategory...	10/06/2005	JCL - info-method
11:28:00:989	/Applications/WebApplic...	10/06/2005	jsp_jcl1128511928940: init
11:27:46:038	/Applications/MyCategory	10/06/2005	Log4j - debug-method with exception
11:27:46:038	/Applications/MyCategory	10/06/2005	Log4j - fatal-method
11:27:46:038	/Applications/MyCategory	10/06/2005	Log4j - error-method
11:27:46:038	/Applications/MyCategory	10/06/2005	Log4j - warn-method
11:27:46:038	/Applications/MyCategory	10/06/2005	Log4j - info-method
11:27:44:536	/Applications/WebApplic...	10/06/2005	jsp_log4j1128511927338: init

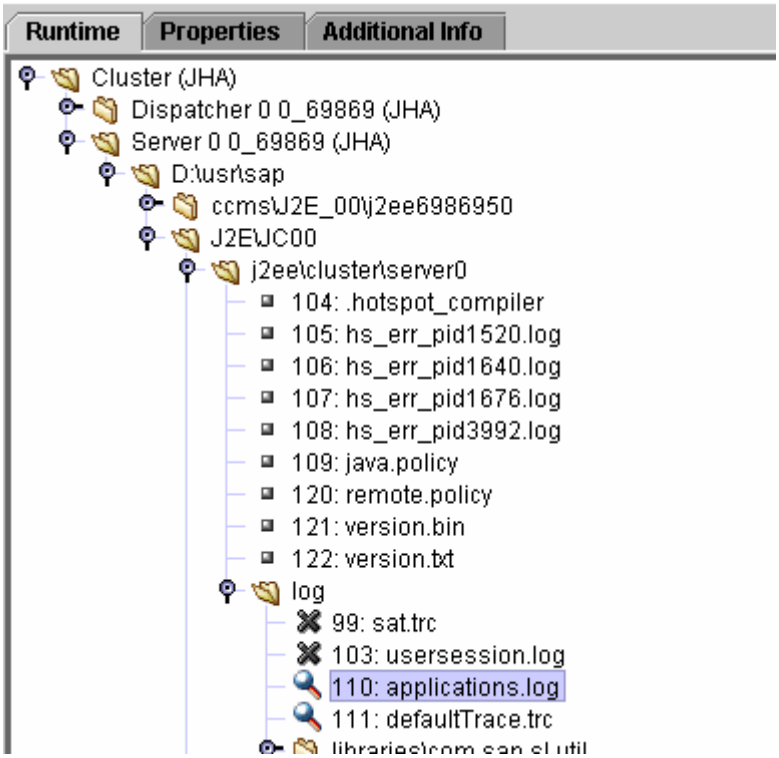
Picture 2 applications.log

The log messages have been successfully routed to the SAP logs.

Running the sample

1. Call the SAP NetWeaver Developer Network using the URL http://sdn.sap.com and log on with your user ID and the corresponding password. If you do not have a user ID, you must register before you can log on.
2. Navigate to *Web Application Server* area and then to the *Samples and Tutorials* section.
3. Download the ZIP file *TutLogging_Integration.zip*
4. Unzip the contents of the ZIP file into the work area of the SAP NetWeaver Developer Studio or in local directory.
5. Call the SAP NetWeaver Developer Studio.
6. Import the sample projects
    a. To do this, choose *File* → *Import* in new menu.
    b. In the next window choose *Multiple Existing Projects into Workspace* and choose *Next* to confirm.
    c. Choose *Browse*, open the folder in which you unzipped the projects *TutLogging_Integration*
    d. Select the projects *TutLogging_Integration_Ear* and *TutLogging_Integration_Web,* check the the *open* checkbox and choose *Finish* to confirm.
7. Build the EAR
    a. Switch into *J2EE Explorer View* in the *J2EE Perspective* in the Developer Studio
    b. In the context menu of the *TutLogging_Integration_Ear* select “Build application archive”
    c. Choose Build Application Archive from Popup
8. Deploy EAR
    a. In the context menu of the EAR File under *TutLogging_Integration_Ear* select “Deploy to J2EE engine”
    b. Choose *Deploy to J2EE Engine*
9. Open the following links in your web browser:
    a. http://<host>:<port>/logging/log4j.jsp
    b. http://<host>:<port>/logging/jcl.jsp
10. Start the *Visual Admin*
11. Connect to your server and login
12. View the logs
    a. Got to Cluster tab
    b. Open the node: *server<your server>/ Services*
    c. Click on the *LogViewer*-Service

d. At the marked position you can view logs and traces



13. By reloading the URLs and Refreshing the log and trace you can see that log messages are completely routed to SAP logging

Using bridges in your project

You are in a project situation where you have to handle different log solutions at once and you accept the restrictions coming along by routing the messages. In case of Log4j all you have to do is:

- Make sure *log4j.jar* your classpath
- Copy the *com.sap.logging.bridge.log4j.SapLogAppender* + *log4j.properties* from the *TutLogging_Integration_Web/source* folder to your project source-folder
- Set your category name in *log4j.properties*
- Modify the *SapLogAppender* to your needs

In case of Jakarta Commons Logging:

- Make sure the commons-logging jars are in your classpath
- Copy the *com.sap.logging.bridge.jcl.SapLogJclImpl* + *commons-logging.properties* from the *TutLogging_Integration_Web/source* folder to your project source-folder
- Set your category name in *SapLogJclImpl*
- Modify the *SapLogJclImpl* to your needs

For all other toolkits this document combined with documentation of your logging solution gives you hopefully enough information to easily write a bridge on your own, that satisfies your needs.

About the author

Johannes Hamel is a technology consultant with main interest on J2EE. He is experienced on different J2EE platforms and different Open Source products. Currently he is focused on the SAP NetWeaver platform as a team member of the Platform Ecosystem – Market Development Engineering for SAP in Walldorf.