

Module 14 sample

Working with the Logging and Discovery components

14.1 Logging makes me happy	1
14.2 Discovery channel.....	6
14.3 Summary	11
Index	12

This module discusses two Commons components that didn't seem to fit anywhere else, for one reason or another. The Logging and Discovery components have been around for quite a while, and they're included together in this last module so we can introduce them with examples; if your interest is piqued, read on for further information. Let's start with the Logging package.

14.1 Logging makes me happy

Logging is one of the silent components of the Commons library. It's almost ubiquitous, makes minimal fuss, and does a lot of good by providing a consistent and flexible logging apparatus.

The Logging component is essentially a wrapper library that works with existing logging implementations. It provides a transparent working environment for writing log statements from within applications. It doesn't provide the logging logic or the log implementation itself: All it does is discover the underlying logging library using a standard mechanism and supply it to your application.

Let's examine the process that the Logging API adopts to look for the underlying libraries.

14.1.1 Searching for the libraries

The Logging component uses a well-defined process to look for the underlying libraries. But what libraries is the Logging component looking for? Most of the popular ones, as listed here:

- Log4J (<http://jakarta.apache.org/log4j>)
- Avalon logkit (<http://avalon.apache.org/logkit/>)
- JDK 1.4 logging
- Its own SimpleLog and NoOpLog

It isn't difficult to add another logging library to this list, which underlines the usefulness of the Logging component. Moving between the underlying library implementations is easy, if Commons Logging is being used as an abstraction; so, it makes sense to use it rather than an implementation.

The task of looking for these libraries falls on the `LogFactoryImpl` class. As the name suggests, this class is the implementation class for the `LogFactory` interface. Just as this factory is responsible for looking for and selecting the right logging implementation, the `Log` interface is responsible for directing logging calls to the underlying libraries. It has several implementations, one for each of the possible logging libraries.

So what is that well-defined process that looks for the underlying libraries? The steps are listed here:

1. The Logging library looks for the value of a configuration attribute named `org.apache.commons.logging.Log` (or `org.apache.commons.logging.log`). Actually, the `LogFactoryImpl` class looks for this attribute, so you can set this attribute in code by calling `LogFactoryImpl.setAttribute(String className)`.
2. If that attribute isn't found, the `LogFactoryImpl` looks for a system property by the same name.
3. If there is no system property by that name, the factory class tries to load Log4J's `Logger` class and its own `Log` implementation for it (`Log4JLogger`).
4. If Log4J's classes can't be loaded, the factory tries to load the JDK 1.4's logging class (`java.util.logging.Logger`) and its own implementation for it (`Jdk14Logger`).
5. If step 4 fails, the factory looks for JDK 1.3's logging class (`java.util.logging.Logger`) and its own implementation (`Jdk13LumberjackLogger`).
6. Finally, if nothing else is available, the factory defaults to using the `SimpleLog` class.

The value of the `org.apache.commons.logging.Log` attribute or the system property from steps 1 and 2 must be equal to the fully qualified name of the corresponding `Log` implementation that is to be used. As of now, it can be any one of the following values:

```
org.apache.commons.logging.impl.AvalonLogger
org.apache.commons.logging.impl.Jdk13LumberjackLogger
org.apache.commons.logging.impl.Jdk14Logger
org.apache.commons.logging.impl.Log4JLogger
org.apache.commons.logging.impl.LogKitLogger
org.apache.commons.logging.impl.NoOpLog
org.apache.commons.logging.impl.SimpleLog
```

The discovery of a library to choose doesn't happen until a call is made to get a `Log` instance to write to. In the next section, you'll see how to make this happen in code.

14.1.2 Starting the process

One of the easiest ways to start using the Logging component is to put `commons-logging.jar` in your application's `CLASSPATH` and let it try to configure itself. As you've seen in the previous section, if all else fails, Logging resorts to using the `SimpleLog` implementation that writes all messages to `System.err`, the standard error output stream (which is equal to the Standard output stream, unless you change it).

But how does the logging begin? When does the factory start looking for the underlying libraries? The first time, a call is made to the factory for a log to write to anywhere in the code. It's as simple as shown in listing 14.1.