# QuickFIXJ Logging

I've had a few discussions with users about logging in QuickFIX/J. There are several issues related to logging than can be confusing. As you may know, QuickFIX/J is based on the QuickFIX C++ JNI API and one of the goals of QuickFIX/J is to be upwardly compatible with the JNI API. The C++ code uses Log and LogFactory abstractions to encapsulate the logging mechanism. This is a reasonable approach, but the design of the abstractions has a few weaknesses.

- There is a Log instance per FIX session, and only session-specific logs
- The Log abstraction does not support levels or priorities on the log messages.</li>
- The default Log implementations are not as flexible as Log4J or the JDK 1.4 logger

The session-oriented Log means that there is no defined way to log information not associated with a session. Some QuickFIX users have recommended having a global log that's not session-specific but this would not be compatible with existing `Log` implementations without some signficant hacks. For example, we could define a pseudo-session identifier and use that for global logging. This seems like a bit of a hack to me.

QuickFIX/J uses the SLF4J log implementation wrapper for logging information not directly related to FIX sessions. This allows Log4J, the JDK 1.4 logger or other logging implementations supported by SLF4J to be used (including a bridge to Jakarta Commons Logging). There is also a QuickFIX/J SLF4J log factory implementation that can be used (with Log4J, for example) to unify logging for both the session and non-session code. For example, you can use Log4J appenders to route logged session events and non-session log information to one file and messages to a database. You could also tell Log4J to route logged event with a priority level above a defined level ("WARN", for example) to a remote socket or a Java Messaging Service queue where the information could be displayed on an operations console or processed by a log aggregator. You could also implement your own custom logging behavior as a plugin to a framework like Log4J. See the description of Log4J appenders for more information on this possibility. These types of advanced logging behaviors are not supported by the default QuickFIX log factory implementations. This was the reason for my comment about flexibility. My recommendation is to use the SLF4J log factory with your favorite logging implementation instead of using the legacy QuickFIX log factory implementations.

The other issue with the QuickFIX log implementations is that session events are logged with no priority specified. This means that you won't be able to use priority-based routing of log messages for session events. The other logged information will have priorities associated with them.

For more information on advanced logging configuration, see the Log4J or JDK 1.4 logging configuration information.

In a future article I will describe some example advanced logging architectures that can be implemented using QuickFIX/J and Log4J.