JA-SIG Conferences : uP101 Vancouver Logging and Log Settings

This page last changed on May 31, 2006 by awp9.

This is a pre-conference seminar element

For the moment, this page is **not** a generic guide to uPortal logging configuration. It is the specific ramblings that will be presented at the JA-SIG Vancouver uPortal 101 seminar. General documentation belongs elsewhere.

This seminar element should take 10 minutes. So Andrew should talk fast!

Why Log?

In which the philosophy, motivation, and intent behind logging is explained.

Logging makes code more maintainable. In debugging a system, crank up the logging and get a readout of some information about its state, what it's doing, how it sees the world. When troubleshooting a slight difference between the behaviors of development and production, diff the log files for sections related to the question. Logging fits somewhere in the continuum between static code inspection and interactive debugging. The right logging can make transparent a system that is too difficult, too locked down to interactively debug, and can capture information even when you're not actively looking.

Logging makes code more literate. Ideally it narrates the activity of the code, drawing attention to critical portions and relevant state, crying out about errors, but importantly just plugging along as an active spectator for your code's work. Some good logging servies in a role of "active comments", it's the same comment you would have written, but now active and including some relevant state.

Logging makes code better designed. Maybe this point is a stretch, but I think it's true: adding quality log messages, articulating what your class does and what its relevant state is, motivates additional thought about what the class does and what its state is. If you're uncomfortable narrating what you're doing, then maybe you're doing something that ideally you don't want to be doing. Sitting down to write the log message that apologizes for your excessively clever trick motivates refactoring towards something more literate.

How does uPortal logging work?

Once upon a time, like every other Java project on the planet, uPortal invented and maintained its own LogService. It was a static singleton that knew how to write to a log file. It was enhanced to be implemented using log4j and to support log levels. This allowed a single universal logging level to be configured for the uPortal instance, and this would control globally how much logging was recorded.

Log4j operates under a metaphor of named loggers, with loggers by convention taking names associated with the fully qualified of the class name of the class doing the logging. This enables advanced configurations in which different components log at different levels. While I am developing the web proxy channel, I might configure logging to include great detail from the web proxy channel while continuing to only log the most important messages from the rest of the framework.

Accessing log4j via the uPortal static LogService prevents taking advantage of these capabilities. All logging via the LogService appears to "come from" the LogService implementation itself.

Fortunately, Jakarta Commons has implemented a thin wrapper Logging API that fulfills many of the requirements that uPortal LogService tried to fulfill. Commons Logging supports log4j as well as other backing logger implementations and is commonly used by uPortal dependencies as well as uPortal itself so skills configuring and using Commons Logging apply universally.

Some logging recipes and motivating examples

Logging and you: using logging to more effectively develop, deploy, and use uPortal