Wonder Logging

This page describes how you should do logging in Wonder as well as in your apps and frameworks based on Wonder. Wonder is a really huge collection of different frameworks so there are currently different ways of logging used in it. Those will be converted to the suggested form when time permits. Below the current and future logging setup are described. For those only interested in the how-do-I-log here the quintessence:

Logging Framework

SLF4J
http://www.slf4j.org/

Needed Frameworks

slf4j-api-1.7.5.jar
slf4j-log4j12-1.7.5.jar

Both are already included in ERJars so no action required in a Wonder project!

Output

The output of SLF4J is redirected to log4j, currently the main log provider in Wonder. That means there is no visible change for you. You keep your log4j configuration, have the same log output pattern etc.

Coding

First you need a logger object:

```
Logger log = LoggerFactory.getLogger(MyClass.class);
```

SLF4J brings you the power of parameterized messages. If you want to log some values you write:

```
log.debug("The value of '{}' is '{}'", key, value);
```

This prevents code concatenation until the logging is really necessary. If the log statement is output all curly braces *{}* are interpreted as placeholders and the given parameters are used in order to construct the message. This means you can safely drop checks like

```
if (log.isDebugEnabled()) { … }
```

making your code less verbose and shorter. Though you should use such a check if you need to pass a computationally expensive parameter:

```
if (log.isDebugEnabled()) {
  log.debug("The value was: {}", computeExpensiveValue());
}
```

The available log levels are matching those of log4j with the exception of the intentionally missing *fatal*:

- ERROR
- WARN
- INFO
- DEBUG
- TRACE

When logging exceptions SLF4J interprets the last parameter as stacktrace if it is an exception object:

```
try {
  …
} catch (Exception e) {
  log.error("Could not do whatever I wanted to do!", e);
}
```

Of course you still can use parameters as long as the exception is the last param:

```
try {
  …
} catch (Exception e) {
  log.error("Error during processing of id {}", id, e);
}
```

Best Practices

Logging normally takes not part of the public API of a class and thus should be kept internally. So your class should neither use class-external logger objects nor provide own logger objects to other classes but declare them in private variables instead. Best practice is to create a logger that way:

```
class MyClass {
  private static final Logger log = LoggerFactory.getLogger(MyClass.class);
  …
}
```

If you want to use a logger associated to another class you can pass that class to the getLogger method. Logger objects are cached so you won't create multiple instances of logger objects for the same class.

More SLF4J info

For more info regarding SLF4J look at the introductory user manual or head to their documentation page.

Wonder Logging Libraries (Current)

Wonder comes with different logging libraries due to its long history and huge number of sources and frameworks. Those are:

NSLog

NSLog is the original logging facility of WebObjects and thus used mainly in the core WO frameworks. With Wonder the logging is configured to redirect its output to log4j (see ERXNSLogLog4jBridge and ERXLogger#configureLogging).

http://wocommunity.org/documents/javadoc/WebObjects/5.4.2/com/webobjects/foundation/NSLog.html

commons logging

The Apache commons logging isn't used by Wonder itself but some of its bundled libraries are relying on it to produce log messages like the Apache HttpClient. You can find the corresponding library in ERJars.

http://commons.apache.org/proper/commons-logging/

log4j

Log4j is the main logging facility of Wonder. For that – as stated above – the logging of WebObjects from NSLog is redirected to log4j so you have only a single logging configuration to maintain. The log4j library is included in ERJars.

http://logging.apache.org/log4j/1.2/

SLF4J

SLF4J is barely used in Wonder currently. It is not a real logging framework like log4j as in fact it acts only as a facade to any logging framework. To produce real logging output you can throw in one of several plugins that redirects the output to a specific logging framework. In Wonder the log4j plugin is used for that, i.e. logging done with slf4j is redirected to log4j as is NSLog. Look at ERJars for both files.

http://www.slf4j.org/

Wonder Logging Libraries (Near Future)

For future development the usage of SLF4J is highly encouraged instead of log4j. As of now all logging done through SLF4J will be automatically redirected to log4j so no configuration change or rewrite is needed by users making the transition seamless.

Why SLF4J

The SLF4J framework has many advantages over log4j:

- *Freedom of choice* – the user can choose which logging framework to use instead of being fixed by Wonder's decision.
- *Performance* – there are many situations where logging is done faster and with smaller memory footprint
- *Code cleanliness* – the API of SLF4J makes it often possible to reduce the code needed for logging

Let's look at the last point and compare. You have a debug statement in log4j:

```
log.debug("The value of '" + key + "' is '" + value + "'");
```

During the method call the compiler will create a StringBuilder to construct the param of the debug method and call the log4j method. That method then will check if the current logger is on the correct level and either output the log statemend or just return.
Now if you wanted to prevent the String concatenation to avoid the parameter construction cost if it is not really needed you would have to write:

```
if (log.isDebugEnabled()) {
  log.debug("The value of '" + key + "' is '" + value + "'");
}
```

With SLF4J there is a cleaner way, parameterized messages:

```
log.debug("The value of '{}' is '{}'", key, value);
```

This defers the parameter construction until it is really needed. Of course if you include a computationally costly parameter you still should check your logging level beforehand:

```
if (log.isDebugEnabled()) {
  log.debug("The value was: {}", computeExpensiveValue());
}
```

Wonder Logging Libraries (Far Future)

The plan is to migrate all logging to use the SLF4J classes and afterwards to remove any SLF4J plugin from Wonder. That is Wonder won't be logging anything out of the box. If you use Wonder you will have to add the SLF4J plugin that *you* want to your project. By this you can control yourself if you want to log via log4j, commons logging, LOGBack or any other available framework.
Wonder could provide some optional logging framework that includes basic configuration and libraries for beginners to setup a simple log setup if they should not be wanting to take this step yet.