



ONIP-3: Allow User Supplied Logging Capabilities

Document status	DRAFT
Document owner	Jeff Zemerick

Introduction

OpenNLP uses `System.out()` and `System.err()` by default for logging. This is fine for instances in which OpenNLP's CLI tools are being used but may not be ideal when OpenNLP is used as a library. Previous work in this area includes

 [OPENNLP-675](#) - Absence of logging and usage of `System.out` . This proposal presents a method for allowing the user to customize the default logging behavior when OpenNLP is used as a library.

Current State

As discussed above, OpenNLP currently (as of 1.7.2) defaults to using `System.out()` and `System.err()` for logging messages and errors, respectively.

The Problem

When using OpenNLP as a library the output of logs to standard out and standard error is not ideal as these logs often need to be captured by the application for external storage and reporting (and also just to keep from cluttering up standard out).

Proposed Solution

The proposed solution is to create an OpenNLP `Logger` interface that developers can implement to customize the logging. The user can provide their own implementation of this interface to control OpenNLP's logging. This interface will have to exist in a new project (perhaps `opennlp-model`?) in order to avoid circular dependencies. (The `opennlp-tools` project will have a dependency on this project. The user's project can either have an explicit dependency on `opennlp-model` or a transitive dependency based on the project's requirements.)



The naming of objects in this proposal is mainly for illustrative purposes. I'm not proposing any specific naming and am open to naming suggestions.

An example of the logging interface is:

```
/**
 * Provides logging capabilities.
 */
public interface Logger {

    void log(String message);
    void error(String message);
    void error(String message, Exception ex);

}
```

The default implementation used when the user has not provided an implementation of `Logger` will be:

```

/**
 * Default implementation of {@link Logger} that logs
 * all messages to standard out and all errors
 * to standard error.
 */
public class DefaultLogger implements Logger {

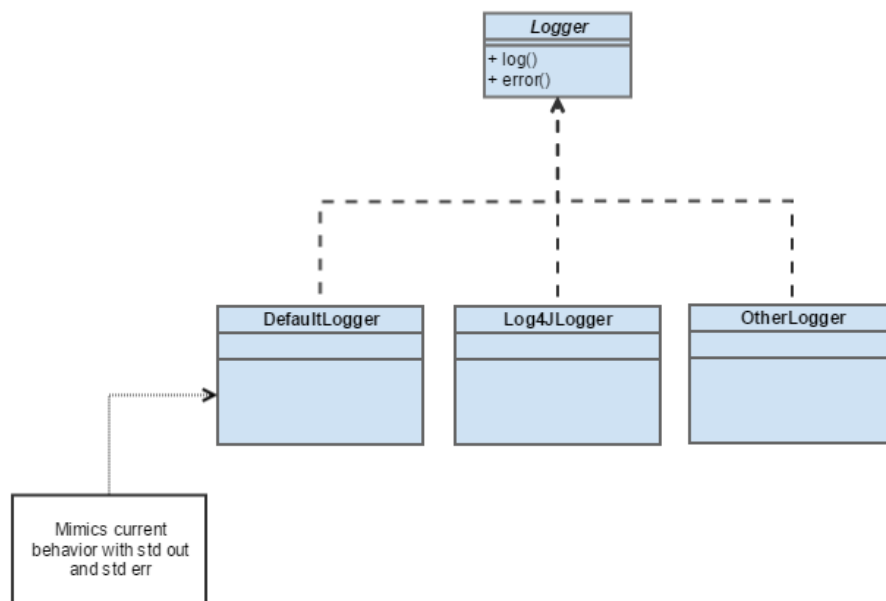
    @Override
    public void log(String message) {
        System.out.println(message);
    }

    @Override
    public void error(String message) {
        System.err.println(message);
    }

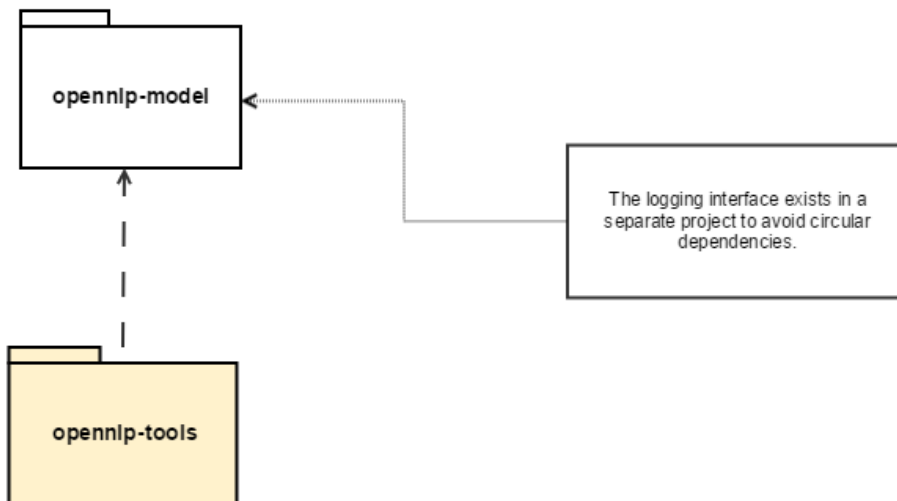
    @Override
    public void error(String message, Exception ex) {
        System.err.println(message);
        System.err.println(ex.toString());
    }
}

```

The `DefaultLogger` class mimics the current behavior of OpenNLP of using `System.out()`.



A new project contains the `Logger` interface to avoid circular dependencies.



A new class in OpenNLP would be created that stores a static reference to the `Logger` implementation. By default this static variable would reference the `DefaultLogger`. The user can set their own `Logger` implementation at any time through the `setLogger()` function.

```

/**
 * Provides access to the {@link Logger}.
 */
public class LoggerConfiguration {

    private static Logger logger = new DefaultLogger();

    public static void setLogger(Logger l) {
        logger = l;
    }

    public static Logger getLogger() {
        return logger;
    }

}
  
```

Existing logging statements in OpenNLP would be modified to perform the logging via the `LoggingConfiguration` class. For example, from [NameSampleCountersStream](#):

Current	Proposed
---------	----------

```

public void printSummary() {

    System.out.println("Training data summary:");

    System.out.println("#Sentences: " +
getSentenceCount());

    System.out.println("#Tokens: " +
getTokenCount());


    int totalNames = 0;

    for (Map.Entry<String, Integer> counter :
getNameCounters().entrySet()) {

        LoggerConfiguration.getLogger().log("#" +
counter.getKey() + " entities: " + counter.
getValue());

        totalNames += counter.getValue();

    }
}

```

```

public void printSummary() {

    LoggerConfiguration.getLogger().log("Training
data summary:");

    LoggerConfiguration.getLogger().log("#Sentences:
" + getSentenceCount());

    LoggerConfiguration.getLogger().log("#Tokens: "
+ getTokenCount());


    int totalNames = 0;

    for (Map.Entry<String, Integer> counter :
getNameCounters().entrySet()) {

        LoggerConfiguration.getLogger().log("#" +
counter.getKey() + " entities: " + counter.
getValue());

        totalNames += counter.getValue();

    }

}

```

Summary

This proposal:

- Presents a way to let users of OpenNLP as a library to control logging.
- Requires:
 - A new project that contains a `Logger` interface.
 - Modifying current `System.out()` and `System.err()` calls to reference the new `LoggerConfiguration` class.

Future Work

A similar approach could be taken for [functions](#) that expect a `PrintStream` object.