Redwood Tutorial

Quick Start

Code

```
import edu.stanford.nlp.util.logging.*

StanfordRedwoodConfiguration.setup();
Redwood.log("Hello World!");
```

Output

```
>> Hello World!
```

Main Ideas

- We use logging to trace code flow
  - Want: Hierarchical structure, etc
  - Less Useful: Log levels / timestamps / etc.
- Single global logger: `Redwood.java`
  - Mark messages with *channels*, e.g. NER, Parser, etc.
- Simple configuration; reasonable default
- Captures existing loggers and `sysout`

Logging: Why do it?

- For your sake:
  - Can filter what you want to see between runs
  - Can revisit logs in log files
  - Structured way to print information
  - Track timing information
- For others' sake:
  - Can filter which components you want to see logs from
  - Can easily track messages not written by you

Logging: Features

- Log Files
- Log Levels
- Log Source (class.method() of call)
- Timestamps

Standard logging frameworks want to track ***events*** *in a long-running program*

Logging: Features we use

- Log Files
- Log Levels
- Log Source (class.method() of call)
  - (but really, want Source Component, not class)
- Timestamps

*We want to track **code flow** in a complex, well structured program*

Logging: Features we add

- Log Files
- Log Levels
- Log Source (component)
- Better show code flow
  - Track *tasks*, not *events*
  - Nested structure
  - Durations, not absolute timestamps
- Collapse repeated messages

Redwood: Setup

- Import Redwood

```
import edu.stanford.nlp.util.logging.Redwood;
```

- Configure Redwood
  - **No configuration** – basic, general framework
  - **Stanford configuration** – tuned to JavaNLP a bit

```
StanfordRedwoodConfiguration.setup();
```

  - **From properties** – most common option

```
StanfordRedwoodConfiguration.apply(Properties p);
```

or

```
RedwoodConfiguration.apply(Properties p);
```

Redwood: Log

Redwood.log([channel]*,[message]);

```
Redwood.log("Hello World!");
   >>                Hello World!
Redwood.log(Redwood.WARN,"warning!");
   >> [WARN]         warning!
Redwood.log(Redwood.WARN,"mytag",annotation);
   >> [WARN,mytag] This is a sentence
```

Idea: *Every message goes through log(); any information goes into channels*

Redwood: Tracks

Idea: We use logs to track execution flow

- A *Track* should encapsulate a task
  - e.g. *Running EM*, *Loading Data*, *etc.*
- Tracks should be nestable: an *outline* of code

```
Running EM {
  Iteration 1 {
    Datum 1: likelihood -2.49
  }
}
```

Redwood: Tracks

Redwood.startTrack([channel]*,[name]);
Redwood.endTrack([matched_name]);

```
Redwood.startTrack("Running EM");
for(int i=0; i<2; i++){
  Redwood.log("EM","Iteration "+i);
}
Redwood.endTrack("Running EM"); //matches startTrack()
  >>                Running EM {
     [EM]             Iteration 1
     [EM]             Iteration 2
                      }
```

Redwood: Setup Revisited

- Configuration uses the builder paradigm
  - `e.s.n.util.logging.RedwoodConfiguration.java`
- Big picture:

```
RedwoodConfiguration
  .current() // or .standard(), .empty()
  .changeInSomeWay()
  .changeInSomeOtherWay()
  .apply(); // ← actually apply config.
```

- Methods documented; add new methods here!

Redwood: Setup Revisited

- Some Methods:
  - **`console()`**: print messages to console (usually already there)
  - **`file(filename)`**: print messages to a file
  - **`loggingClass(classname)`**: ignore this class when finding the source of a log message
  - **`neatExit()`**: stop Redwood cleanly on an unexpected exit
  - **`static parse(properties)`**: parse a properties object (documented in JavaDoc)
  - *...and more!*

Redwood: Log Revisited

- Magic with channels
  - `Redwood.{ERR,WARN,...}`: Log levels
  - `Redwood.FORCE`: Print me no matter what
  - "`str`": adds this message to the "str" channel
- Utilities
  - Redwood.Util has a bunch of helper methods

    tip: `import static Redwood.Util.*`

  - For logf (printf syntax) with channels:

    `Redwood.channels([channels]*).logf("%s","hi");`

Redwood: Message Handlers

- Tree of Message Handlers, e.g.
  - Visibility Handler (console)
    - Repeated Message Handler (console)
      - Console Handler
  - Visibility Handler (file)
    - Repeated Message Handler (file)
      - File Handler
- Make your own: extend `LogRecordHandler`
- Add a handler:

```
RedwoodConfiguration.current()
    .handler([parent], [handler_to_add])
    .apply();
```

Redwood: Visibility Handler

- Show only a subset of channels
  - e.g. only “EM,” or only WARN and ERR

- Common enough for top-level methods:

```
Redwood.hideAllChannels();
Redwood.showOnlyChannels([channels]*);
Redwood.showAllChannels();
Redwood.hideOnlyChannels([channels]*);
```

Advanced: Repeated Records

- **Use Case:** Same [generic] error prints often
- **Solution:** Collapse identical records

```
RedwoodConfiguration.current()
     .collapseExact().apply();
for(int i=0; i<435; i++){
  Redwood.log("Generic message")
}

>>          Generic message
>>          (last message repeated 434 times)
```

Advanced: Repeated Records

- **Use Case:** Similar message prints often
- **Solution:** Collapse similar records

```
RedwoodConfiguration.current()
    .collapseApproximate().apply();
for(int i=0; i<40; i++){
  Redwood.log("Iter "+i+": logZ = -175.9")
}
```

```
>>          Iter 1: logZ = -175.9
>>          Iter 2: logZ = -175.9
>>          Iter 3: logZ = -175.9
>>          Iter 24: logZ = -175.9  //print every sec.
>>          ...36 similar messages
```

Extra: Colors!

- For track names:
  - For random track colors, try setting the property
    `log.{console,file}.colorChannels`
  - For specific colors, try setting the properties
    `log.{console,file}.track[Name]Color`
    `(e.g. log.console.trackParserColor = Blue;)`
  - Also:
    `log.{console,file}.track[Name]Style`

- For coloring output, pass in a color (`e.s.n.util.logging.Color`) as a channel.

Extra: All your loggers are belong to us

- Capture `stdout/stderr`:
  - Redirects standard system streams to Redwood's `STDOUT` and `STDERR` channels
  - Default in `StanfordRedwoodConfiguration.setup()`
  - Also: `RedwoodConfiguration.captureStreams()`

- Capture java.util.logging:
  - Redirects calls to Java's logger to Redwood
  - SEVERE → ERR, WARN → WARN, <= FINE → DEBUG
  - Default in `StanfordRedwoodConfiguration.setup()`

Thank You!

Code

```
import edu.stanford.nlp.util.logging.*

StanfordRedwoodConfiguration.setup();
Redwood.log("Hello World!");
```

Output

```
>> Hello World!
```