clad

# TODO

* Figure out how to add a unit test that makes sure a command is identified and loaded from a JAR.
* Currently we look for classes based on their name (\*Command and \*ApplicationRuntime). Use annotations or look inside the class for to see if they implement Command or ApplicationRuntime.
  + In order to find the ApplicationRuntime, we need to set “application.name” system property. This is awkward, we need to get rid of it. Annotations is probably best.
* ApplicationRuntime callback should be annotated instead of being interface methods.
* Mechanism to link cliff to its clients – where to find the commands. Most likely a package or a naming convention.
* The convention for a default command if none is specified.
* Define how users can extend their configuration – the user application will most likely need global options of their own
* Implement –s--somenting=something argument convention.
* The Help – initially based on a flat text file in the client.

# Overview

<wrapper> [global-options] command <command-options> sub-command <sub-command-options>

Example:

events --format=”something ….” sample –s 1

Each command line option has a configuration file correspondent. Command line value takes precedence over the configuration file value.

The corresponding configuration file:

format: something

sample:

option-1: value-1

option-2: value-2

# How does it Work?

The framework scans the command line looking for the first command.

Everything between the wrapper name and the command name is interpreted as global option.

# Command Name Syntax

# Usage

Implement ApplicationRuntime

Implement Command(s).

Package the applicationRuntime and the commands in a JAR (or place them in a directory).

Set “application.name” as a system property. If the applicationRuntime is BlahApplicationRuntime, the application.name must be “blah”.

Make sure the JAR or the directory is first on the class path (otherwise other <your-command-name>Command.class, if exist, will be instantiated first)