clad

# Tactical TODO

* Define how users can extend their configuration – the user application will most likely need global options of their own.
  + An application must declare its global options so we know the literals (short or long).

# Long Term TODO

* 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. ApplicationRuntime callback should be annotated instead of being interface methods. However, keep the framework backward compatible, old API users should keep working.
  + 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.
* Define a configurable convention for a default command if none is specified, if I feel the need. Currently is hardcoded in the application runtime: ApplicationRuntime.getDefaultCommandName(). It’s probably sufficient.

# 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.

It uses the GNU command line convention –o <value> | --option=value.

# How does it Work?

The framework scans the command line looking for the first argument that can be mapped to a command. Mapping to a command means scanning the classpath and looking for Command interface implementations whose simple class name match the following pattern: <commandName>Command.

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

Everything beyond the command name is interpreted as command’s option.

# Usage

## ApplicationRuntime

Implement ApplicationRuntime

## Command

Implement Command(s).

If a specific command does not need an application runtime instance (thus the framework must not instantiate an application runtime for it), implement the Command so its needsRuntime() returns false. By default CommandBase.needsRuntime() returns true.

Add <command-name>.txt containing the command help in the same package as the command class.

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).

## Command Execution

execute() will be called on the main thread.

# Help

## Application Help

The application has a generic help that is displayed every time the “help” without any arguments is invoked. The help content must be placed in a text file in the same package as the ApplicationRuntime implementation class. The name of the help file must be <application-name>.txt.

The help renderer recognizes several macros, which are replaced by dynamically generated content at runtime.

Macros:

@COMMANDS@ - inserts the list of commands available to the application. The runtime builds that list via introspection looking for classes that implement the Command interface.

## Command Help

Each command could specify its individual help content as a text file placed in the same directory as the class file. The name of the file is <command>.txt.

The help is invoked with any of “help”, “--help” and “-h”

# Verbose Option

-v or --verbose turns on DEBUG on the underlying CONSOLE appender.

# Configuration File

The corresponding configuration file:

format: something

sample:

option-1: value-1

option-2: value-2

# Default Command

If no command is specified, the framework will use the “default command”, if there is one. If not, the application should display:

[error]: no command specified on command line and no default command was configured.

Instructions on how to configure the default command.