jim.schmidt@dbexperts.com

April 7, 2010

Abstract

This article introduces a simple mechanism for declaratively creating a robust command line interface. This mechanism extends the functionality of the SourceForge jcmdline project available at http://jcmdline.sourceforge.net/ by providing declative arguments and options functionality using java annotations.

Argument fields must be objects, not primitives and the setter must be the same as the type for the field, no autoboxing

1 Introduction

The javautil-commandline contains classes used to parse and process command line options and arguments from a Java program. This package was written with the following goals:

- support option and argument processing described declarative fashion using java annotations in a java argument bean class
- support locale specific messages for command line processes using resource bundles
- Support the declaration, description and help displays for command line options in a declarative fashion using Java Annotations.
- Encourage uniformity of command line conventions and usage display throughout a project.
- Make it difficult to add a command line option without its being documented in the command usage.

1.1 Features

- Parses command line options and arguments.
- Supports frequently used parameter types - booleans, strings, integers, dates, file names...
- Performs common validation tasks - checks for number ranges, file/directory attributes, values that must be from a specified set, etc..
- Displays neatly formatted usage and error message in response to parameter specification errors.
- Provides support for commonly used command line options (such as -help and -version) and a means to standardize command line options across all executables of a project.
- Supports hidden options.
- Create Command Line Switch Processing with very little effort
  - Create a class (a bean) for the arguments
  - Annotate the fields of the bean to indicate
    - ∗ is this getting too deep?

Create an annotated argument bean and a corresponding properties file.

2 The Argument Bean

	Boolean
	Date
Supported argument types	File
	hline Integer
	hline String
	hline

3 The properties file

DirectoryExists
DirectoryReadable
DirectoryWritable
Exclusive beginverbatim // not annotated should not be set private String parts; @Exclusive(property=”parts”) private String bit endverbatim
FileExists
FileReadable
FileWritable
Multivalue
Optional
Required
RequiredBy
RequiredUnless

3.1 ArgumentBean

Write a bean with objects and accessors.

Annotate each argument with either @Optional or @Required. Supported Object types are

- File
- Integer
- Date

3.2 Properties

4 Background

4.1 Command Line Switches

Command line switches were first made popular with the wide acceptance of Unix.

4.2 GNU Get Args

4.3 jcmdline

5 Terminology

Option

Argument

6 jcmdline

6.1 Overview

The jcmdline package contains classes used to parse and process command line options and arguments from a Java program. This package was written with the following goals:

- Simplify processing of command line options and parameters.
- Encourage uniformity of command line conventions and usage display throughout a project.
- Make it difficult to add a command line option without its being documented in the command usage.

6.2 Features

- Parses command line options and arguments.
- Supports frequently used parameter types - booleans, strings, integers, dates, file names...
- Performs common validation tasks - checks for number ranges, file/directory attributes, values that must be from a specified set, etc..
- Displays neatly formatted usage and error message in response to parameter specification errors.
- Provides support for commonly used command line options (such as -help and -version) and a means to standardize command line options across all executables of a project.
- Supports hidden options.

option A command line option is comprised of an identifying ”tag”, typically preceded by a dash or two, and optionally requires a value. A typical option might be ”-outfile /tmp/myfile”.

argument A command line argument is specified on the command line after all of the options and their values.

parameter A command line parameter is an option or an argument.

6.3 Parameter

6.3.1 tag

The tag is used to identify the parameter.

For options, the tag indicates how the option is specified, for instance, a tag of ”outfile” indicates an option specified like ”-outfile /tmp/myfile”.

For command line arguments, the tag is typically used by the usage display formatter to identify the argument.

6.3.2 description

The description is used by the usage display formatter to describe a parameter. optional indicator The optional indicator indicates whether a parameter is optional or required. If a parameter is configured to be required, the CmdLineHandler will reject the command line if the parameter is not specified.

6.3.3 multi-valued

A parameter that is multi-valued can be specified more than once. This is frequently used with the final command line argument when the command accepts multiple instances. The above example demonstrates this type of usage. acceptable values Most Parameter types allow the programmer to specify a set of values that the parameter will accept. The command line is rejected if a user attempts to set the parameter to any non-specified value.

6.4 hidden indicator

The hidden indicator is used to indicate whether a parameter is hidden or public. By default, all parameters are public, meaning that they will be described in a regular usage statement.

Optionally, a parameter may be designated hidden, in which case it may still be specified by the user, but is not displayed in a normal usage statement. Who has not shipped code supporting ”debug” or ”trace” options that might be performance impacting and thus not be suitable for publication to the casual user? This indicator is for that type of option.

Both the DefaultCmdLineHandler and HelpCmdLineHandler support (hidden) command line options that will cause hidden options and help to be displayed.

6.5 Help

6.6 Option Types

6.6.1 boolean

boolean (true / false) switch

6.6.2 File

Handles filename parameters

6.6.3 Int

Numeric Integer Parameters

6.6.4 DateTime

todo Not supported at this time.

6.6.5 Date

6.6.6 Time

6.7 MultiValued

If an option may be specified multiple times use the emphMultiValue tag to annotate a Collection of the as in

```
@Optional
@MultiValue
private Collection<String>words;
```

7 Cookbook

7.1 The Argument Bean

An ArgumentBean is any class that contains the variables to be set based on command line arguments augmented by annotations that specify the properties (see 6.6 on page 4)

7.2 Declare The Variables

7.3 Create the Resource File

8 Features

8.1 Multi Language Support

Different Resource bundles per language.

8.2 Messages and Help

9 Notes

Every field in the argument class must have an annotation. Why, I have no idea. but this isn't true for booleans??

Every argument must be specified in resource bundle. TODO is there a way to ensure that it is present in every locale and to show the messages for every locale? describe ResourceTag

Exception in thread "Main Thread" java.lang.IllegalStateException: field words of class commandline.CommandLineArguments must have a valid com.custdata.util.cmdline.annotations annotation assigned to it

Must be one of optional or mandatory, should just default to optional

10 Argument Bean

An argument bean is a class of Objects that hold the arguments used by the program.

11 Code

11.1 Optional

```
package org.javautil.commandline;

import org.javautil.commandline.annotations.Optional;

public class IntegerArguments {

@Optional
private int intValue;

public int getIntValue() {
return intValue;
}

public void setIntValue(int intValue) {
this.intValue = intValue;
}

public void evaluateArguments(String[] args) {
CommandLineHandler clh = new CommandLineHandler(this);
clh.evaluateArguments(args);
}
}
```