Command Line Utility 1.0 Component Specification

1. Design

The Command Line Utility simplifies the process of managing command line flags and parameters within Java applications. To use the Command Line Utility, the application specifies the flags (switches) that it wishes to support. The CLU can be configured to validate each argument. The CLU provides a few simple validators, and applications can provide their own validation. Because custom validators are executed within the application context, they can be used to check information available only at runtime.

Terminology

Switch: A command line flag starting with the switch delimiter (usually '-' for UNIX and '/' for Windows). Each switch can accept a minimum and maximum number of values, and can be specified as required.

Value: An argument to a switch. Mulitple values can be separated by commas (or another separator specified at run time).

Parameter: A command line value not constrained by a switch. All parameters must follow all switches.

Example:

```
java app -switch value -switch2 value1, value2 param1 param2
```

This should be parsed as: a Switch named "switch" with one value, a Switch named "switch2" with two values, and two parameters named "param1" and "param2".

The same command line could also be written as:

```
java app -switchvalue -switch2=value1, value2 param1 param2
```

1.1 Design Patterns

The Command Line Utility uses a Strategy pattern to separate validation logic from switch parsing logic. The Validator is an abstract Strategy that can then be subclassed for specific validation algorithms.

Note: The Abstract Factory pattern does not apply for Validators because it would require modifying the Factory for each new Validator. However, the calling application may wish to use an Abstract Factory to manage its own Validators.

1.2 Industry Standards

None.

1.3 Required Algorithms

The parse() method of CommandLineUtility needs to parse an array of Strings containing the arguments passed to the main method of an application.

Because the operating system will separate the command line into Strings based upon spaces in the input, some switches may have arguments in multiple Strings in the input array.

The values for each switch specified on the command line should be passed to the appropriate Switch object provided by the calling application for validation. At the end of

parsing, the validSwitches List should be populated with the switches that were successfully validated during processing. The invalidSwitches List should be populated with switches that were not successfully validated. A UsageException should be thrown if parsing fails.

Variable Length Switches

Some switches may be able to accept a variable number of arguments. This means that it may not be possible to distinguish a subsequent argument from a new switch. Two methods may be used to clarify the situation.

1. All arguments that begin with the switch delimiter are switches; all arguments that do not begin with the switch delimiter are parameters or arguments.

2. All multi-valued switches must use a separator to provide multiple values (for instance –multi a, b, c uses a comma to separate its three values – which happen to appear in different Strings in the input array).

Both methods are required. The first is necessary in cases where a switch can have 0 or 1 argument to distinguish whether the next token is an argument or a switch (e.g. –variable –switch should be interpreted as two switches; -variable value should be interpreted as one).

In general, end-users should be encouraged to use a separator to provide multiple values, as this is unambiguous. A blank (zero-length) argument can be specified this way (adjacent separators).

Pseudocode for parse()

For each argument

If begins with switch delimiter

Get relevant switch

If multiple values allowed

Get subsequent arguments as necessary

If single value allowed

Get next argument

Pass values to Switch object and validate

If valid, add to valid list

If invalid, add to invalid list

1.4 Component Class Overview

ArgumentValidator
Abstract class for argument validation. Subclasses throw ArgumentValidationExceptions to indicate validation failures.

CommandLineUtility
The primary class in this component. Parses the command line using provided Switch objects; throws UsageExceptions when parsing fails.

EnumValidator
Validator for enumerated list switches. Subclass of ArgumentValidator.

IntegerValidator
Validator for integer switches. Subclass of ArgumentValidator.

Switch

Defines a command line switch; holds switch value(s) after parsing.

functionaltests.DelimiterTests
Unit tests for alternate delimiters.

functionaltests.EnumTests
Unit tests for EnumValidator.

functionaltests .FunctionalTests
Wrapper for functional unit tests.

functionaltests.IntegerTests
Unit tests for IntegerValidator.

functionaltests.MultipleValueTests
Unit tests for multiple value switches.

functionaltests.ParameterTests
Unit tests for parameters.

functionaltests.SingleValueTests
Unit tests for single value switches.

1.5 Component Exception Definitions

ArgumentValidationException
Thrown by an ArgumentValidator subclass when validation fails. Provides detailed information about the invalid argument.

UsageException
Thrown by CommandLineUtility when command line is invalid or cannot be parsed.

InvalidSwitchException
Thrown by CommandLineUtility when a switch has a duplicate name or uses invalid characters (for instance, a delimiter); thrown by Switch when the minimum number of arguments is greater than the maximum.

2. Environment Requirements

2.1 TopCoder Software Components:

None.

2.2 Third Party Components:

None.

3. Installation and Configuration

3.1 Package Name

com.topcoder.util.commandline

3.2 Configuration Parameters

None.

3.3 Dependencies Configuration

None.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Execute ‘ant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

To use the CommandLineUtility, create a Switch object for each switch that the application accepts. Pass the switches to a CommandLineUtility object. Call the parse() method of CommandLineUtility with the array of command line arguments.

To add custom validation routines, subclass ArgumentValidator and add a Validator subclass to each appropriate Switch.

See test cases and diagrams for more information.

4.3 Demo

N/a. See unit tests.

4.4 Examples

To print usage for an initialized CommandLineUtility object *clu*:

```
List switches = clu.getSwitches() ;
for (Iterator it = switches.iterator() ; it.hasNext() ; ) {
      Switch s = (Switch) it.next() ;
      System.out.println("\t" + s.getName() + ": ")
      System.out.println(s.getUsage()) ;
}
```