

Module 12 sample

Command-line processing with CLI

12.1 The process of command-line interface.....	1
12.2 Introducing CLI	5
12.3 Summary	25
Index	26

Applications don't work in isolation. A simple enough statement, which means that any application interacts with its environment, affecting the way it behaves. This interface to an application lets you modify application parameters that affect the application's behavior, either at start-time or at runtime. In this module, we'll look at the Command Line Interface (CLI) component from the Commons stable, which enables optimal processing of application parameters at an application's start-time. (The previous module on Modeler discussed how you can manage application parameters at runtime.)

We'll first look at command-line processing. It might seem that the process is simple, but in reality it involves three stages, as used by CLI. We'll then take a brief look at the CLI API. This will set us up for the CLI examples that follow.

12.1 Command-line processing

Command-line processing is the means by which an application understands its initial environment. It sets the stage for subsequent processing. The process that creates the application supplies the parameters—or, as they're known in CLI, the *options*—that define this environment. The application then parses these options and executes accordingly.

As defined by CLI, there are three stages in the processing of options on a command line for an application: definition, parsing, and interrogation. Figure 12.1 shows these three stages.

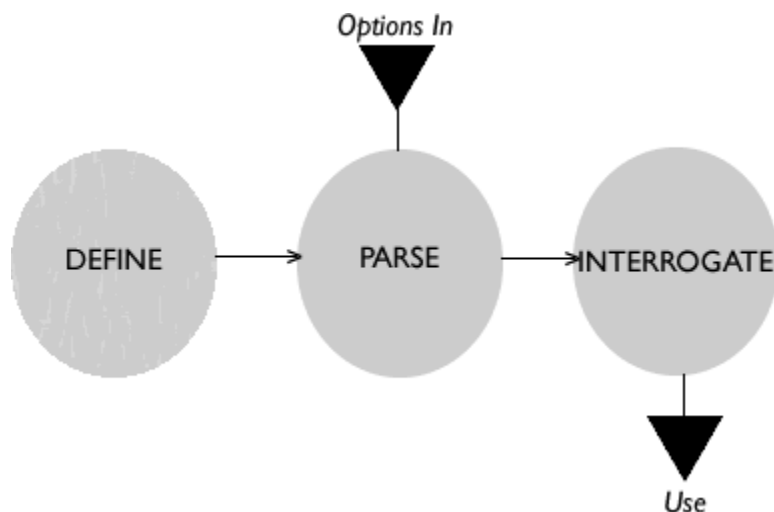


Figure 12.1 The three stages of command-line processing

We'll introduce each of these stages with some examples.

12.1.2 Defining the options

Option processing starts with a definition of each option for the application. This implies that you're required to know in advance all the options possible for an application. This makes inherent sense, because the application itself needs to be aware of the environment and the factors that affect it. You can define options as part of the main application code or as a helper class that creates these options for you. By choosing the latter approach, you can separate application logic from option-creation tasks and let the main application code query the helper classes for the presence or absence of options.

Listing 12.1 shows a very simple application that uses a helper class to define and parse an option. The application code requires that the user be welcomed with a personalized greeting. The application uses a helper class not only to define the user's name as an option, but also to validate and return it.

Listing 12.1 Option definition using a helper class

```
package com.manning.common.chapter12;

public class MainApplicationV1 {
    public static void main(String args[]) {
        System.err.println("Hello " + HelperV1.processArgs(args));
    }
}

class HelperV1 {
    static String processArgs(String args[]) {
        if(args.length != 2 || !args[0].equals("-n")) { ← Define options
            usage();
            System.exit(-1);
        }

        return args[1];
    }

    static void usage() {
        System.err.println("java MainApplicationV1 -n [Your Name]");
    }
}
```

As you may notice, the helper class defines options at the same time it's validating them. For simple applications like this, option definition is almost always done with option validation. Here, the option definition declares that two arguments are required in order to successfully run the `MainApplicationV1` application and that one of them must be `-n`.

As you'll see later, CLI makes the process of option definition more formal by using the `Option` and `Options` classes. Next, we'll look at how to parse options and make them available for use.

12.1.2 Parsing options

Option parsing is the step that accepts, separates, and prepares the options on a command line as defined in the previous stage, for processing by the application code. In listing 12.1, options are parsed in the