clp Reference

Willis L. Boughton

September 2026

---

clp is a Java package for defining and executing text commands in a Java program. Commands can be entered manually or put in command files, which can call other command files. An example command might be "calc:best datasets:one,two,three print". clp has built-in (base) commands, and an application can add any number of application-specific commands. Commands can have defined options that can be of defined types. A command can have execution preconditions, and clp maintains execution state variables and execution logging. clp provides only a sequence control structure, not selections or loops.

There are two types of clp commands, Gui and Console. Gui commands can be entered and displayed only in a clp Java Swing program. Most Console commands can be entered and displayed in Swing program or an OS console program. With base commands alone, a clp GUI program can have panels of GUI buttons that execute commands. *Overall, clp provides a framework for executing Java programs, console or Swing, by means of command files, thereby providing "scripting" of a Java program.*

License

clp is distributed as freeware. It can be used on any number of computers by any number of users and may be distributed freely. clp is provided "as is", and any warranties of any kind, express or implied, for any use or intended use, on any computing system, are disclaimed, including, but not limited to, implied warranties of merchantability and fitness for a particular purpose. In no event shall the software provider be liable for any damages of any form arising in any way out of the use of this software, even if advised of the possibility of such damage. By using clp, you acknowledge that you have read these terms, understand them, and agree to them.

Distribution

clp is distributed in Zip file clp-distrib.zip, which contains the following:

- JAR file clp.jar, a non-executable JAR containing all clp class files. clp.jar must be in the Java classpath for any clp program.
- Package clp.demo, which contains documented, demonstration source code and executable JAR files for the three programs demonstrated in the tutorial video:

    1) Program clp.demo.ConsoleProg, which demonstrates how to make a clp console (command-line) program from clp.AbstractConsoleProg. The executable JAR file for this program is console-prog.jar.

    2) Program clp.demo.GuiProg, which demonstrates how to make a clp GUI program from clp.AbstractGuiProg. The executable JAR file is gui-prog.jar.

3) Program `clp.demo.CustomGuiProg`, which demonstrates how to use the `clp` API to create a custom user interface for `clp.AbstractGuiProg`. The executable JAR file is `custom-prog.jar`

See the `readme.txt` file in package `clp.demo`, and see the `clp` API javadoc at www.bluemontsw.com/clp/api.

Getting Started

To get started with `clp`:

- Watch the "quick look" video on the Web site.
- Watch the tutorial video on the Web site.
- Study this document.
- Study and run the three example programs described above, and study the javadoc.

Command Grammar

The `clp` command grammar is:

```
letter = (isJavaLetterOrDigit is true)
digit = (isJavaDigit is true)
command = argument { ' '  argument }
argument = name [ ':' value ]
name = letter { letter | digit }
value = item { ',' item }
item = ["] { number | name } ["]
number = integer | real
integer = digit { digit }
real = integer '.' integer
comment = '//' ...
```

- All keywords are case insensitive.
- Each command must be on a separate line.
- Arguments can be in any order and be required or optional.
- Each item value has a defined, required type:

  * indicates a value that can be any text without spaces.
  # indicates a value that must be a number. Numbers are handled in floating-point.
  > indicates a discrete value in a set of possible values.
  @ indicates a value that is the name of an execution state parameter.

Class Summary

Table 1 summarizes `clp` classes. All classes are in package `clp`. and italics denotes an abstract class. *In this document, `clp` denotes the `clp` package, and `Clp` denotes the `Clp` class.*

Table 1. Classes	
*Name**	*Description*
`Clp`	A command-line processor for executing a sequence of text commands, one per line, case insensitive. Every `Clp` has built-in (base) commands, and any number of commands can be added.
`Clp.State`	A `Clp` state consisting of state parameters.
`ClpCmd`	A `Clp` command in the `clp` command grammar.
`ClpCmd.Console`	A `Clp` command that uses only an OS command-line window. Any `Clp` can have any number of `ClpCmd.Console` commands.
`ClpCmd.Gui`	`A Clp` command that uses a Swing GUI. A `Clp` that uses Java Swing can have any number of `ClpCmd.Gui` commands as well as any number of `ClpCmd.Console` commands.
`ClpCmd.Arg`	A `ClpCmd` argument. A `ClpCmd` can have any number of arugments.
`ClpCmd.Option`	A `ClpCmd` argument option. A `ClpCmd` argument can have any number of options.
`ClpCmd.Precondition`	A `ClpCmd` pre-condition that must be true for the command to execute. A `ClpCmd` can have any number of pre-conditions.
`CmdFileButton`*	A Swing `JButton` subclass that executes a `clp` command file and manages button interaction.
`CmdPanel`*	A Swing `JPanel` subclass containing a command logging area at the top and a command entry field at the bottom.
`DefaultAppPanel`*	A Swing `JPanel` subclass that is the default application panel for a `clp Gui` program. See section "Making a Gui Program."
* Used only for a `clp` GUI program.	

Program Types

There are two types of `clp` programs, corresponding to the two classes of `ClpCmd` commands:

- A `clp Console` program uses a `Clp` that has only `ClpCmd.Console` commands. A `clp Console` program uses only the OS command-line window.
- A `clp Gui` program uses a `Clp` that has `ClpCmd.Console` commands and, optionally, `ClpCmd.Gui` commands. A `clp Gui` program is a Java Swing program.

Base Console Commands

Table 2 describes the base (built-in) `clp Console` commands. All built-in commands begin with `$`. `[]` indicates an optional command argument.

Table 2. Base Console Commands	
The following commands can be executed in a clp Console or clp Gui program.	
Syntax	Function
`$comments:|on|off|`	Turns listing of command file comments on and off. The default is `off`.
`$date`	Lists the current date.
`$echo:|on|off|`	Turns echoing of keyboard-entered commands on and off. The default is `off`.
`$end:|this|all|`	Ends the `clp`. If the selected option is `this`, the `clp` ends only if the `$noends` state parameter value is zero. (See command `$noend` and section "Ending a `clp`".) If `all`, the `clp` ends regardless of the `$noends` value.
`$fileoutput:|on|off|`	Controls listing of commands and command results in a command file. If `on`, commands and command results, if any, are listed. If `off`, only the entered command file name is listed, even if it calls other command files. The default is `on`.
`$help:|*|all|base|app |console|gui| [precond]`	Lists available commands. If the selected option is `all`, all commands are listed; if `base`, only base commands; if `app`, only application-specific commands; if `console`, only `Console` commands; and if `gui`, only `Gui` commands. Otherwise, the only commands listed are those that contain the * string entered. If `precond` is specified, the preconditions for each command are listed also.
`$list:|*|all|`	Lists command files that are in the root folder specified by state parameter `$rootfolder`, including children of the root folder. If the option entered is `all`, all files are listed. Otherwise, the only files listed are those whose name contains the * string entered.
`$logfile:|on|off|`	Turns on and off the logging of commands and command results to the logging file `clplog.txt`. The default in `on`. See the "File Logging" section.
`$noend:|push|pop|show|`	Controls how a `$end` command is handled. If the selected option is `push`, the `$noend` state parameter is incremented by one. If `pop`, it is decremented by one, to a minimum of zero. If `show`, the $`noend` state parameter is listed. This command can be used to control execution of command files that call other command files. See also section "Ending a `clp`".

$parm:|*,*|*| [option: |remove|save|restore|]	Sets, removes, saves, or restores a clp state parameter. If two values are specified (*,*), a new state parameter is created. The first value is the state parameter name and the second is the parameter's value. If only one value is specified (*) and the option argument is not specified, the parameter is created with value "no value" if it does not exist. If the remove option of argument option is specified, the state parameter is removed. If the save option is specified, the parameter is saved so that it can be restored. If the restore option is specified, the parameter is restored. save and restore apply only to Java String parameters.
$prefix:|rootpath |fullpath|	Sets the prefix for listing of commands in a command file. If the selected option is rootpath, the prefix is the command file's path relative to state parameter $rootfolder. If fullpath, the prefix is the full path of the command file. The default is rootpath.
$text:*	Lists the specified text, wrapping at the length specified by the $textlen state parameter
$textfile:*	Lists a text file word by word, wrapping as for the $text command.
$textlen:#	Sets the maximum displayed line length for the $text and $textfile commands. The range is 40 to 120; the default is 80.
$state [reset:|all|app|	Lists the state parameters. If the optional argument reset is not supplied, the state parameters are listed. If it is supplied with option all, all state parameters are reset; with option app, only application-specific parameters are reset. See method resetAppParms() in the API.
$type:*	Lists a command file, line by line.
$version	Lists the clp version. The default value is the date of the clp base commands. An application can set its version.
The following commands can be executed only in a clp Console program.	
Syntax	Function
$autodelay:|on|off|	Turns automatic delay on and off. If on, the clp delays after each command. The delay length is the value specified with the $delay command.
$delay [set:#]	If set is not specified, executes the set delay; otherwise, sets the delay in seconds. The default value is 5.

Base Gui Commands

Table 3 describes the base (built-in) clp Gui commands. These commands are available only in a Swing program subclassed from Clp.AbstractGuiProg, which uses one Swing JPanel subclass for all application GUI components. This panel is referred to as the "application" panel. The default application panel is an object of class Clp.DefaultAppPanel. See section "Making a GUI Program".

Table 3. Base Gui Commands	
The following commands can be executed only in a Gui program.	
Syntax	Function

`$appanel:#`	Sets the fractional screen size of the output panel of `Clp.DefaultAppPanel`. The value must be between 20 and 80 (percentages). The default is 67.
`$button:* cmdfile:* [hover:*] [color:#,#,#]`	Creates a control button in the most-recently created button row. The `button` * option specified the button's label, and the `cmdfile` * option specifies the name of the command file to be executed when the button is clicked. The `hover` * option specifies the hover text for the button. The `color` argument specifies an RGB background color for the button.
`$buttonrow:* [color:#,#,#]`	Creates a row (`JPanel`) for control buttons. This row is positioned at the bottom of the application panel. The value of the * option specifies the row's name. The `color` argument specifies an RGB background color for the panel. There can be any number of button rows.
`$clearlog`	Clears the command logging panel.
`$clickbutton:*`	Clicks the button with the label specified by the * option. This action is exactly as if the user had clicked the button.
`$nobuttons`	Removes all buttons and button rows.
`$view:|log|apppanel|`	If `log`, sets the `Clp.AbstractGuiProg` window view to show the logging view. If `appanel`, sets the window view to show the application panel. See section "Making a clp GUI Program".

Execution State

A `Clp` maintains execution state parameters. The default parameters are:

- `#delay`, whose value is set by the `$delay` command
- `#echo`, whose value is set by the `$echo` command
- `#fileoutput`, whose value is set by the `$fileoutput` command
- `#logfile`, whose value is set by the `$logfile` command
- `$noends`, whose value is the current number of pushed noends
- `#prefix`, whose value is set by the `$prefix` command,
- `$rootfolder`, whose value is a `clp` program startup argument, and
- `#textlen`, whose value is set by the `$textlen` command.

Parameters whose names begin with `$` can not be changed or removed by the user. Parameters that begin with `#` can not be removed but can be changed by the user with the appropriate base command. With the `$parm` command, an application can add any number of state parameters, with names that do not begin

with $ or #. With the $state command, the value of a state parameter can only be a string. With the clp API, the value of a state parameter can be any object.

The clp state class is Clp.State. (See the API.) An application can subclass Clp.State to obtain an application-specific state. Though "state" normally is understood as only "variables" not functionality, an application-specific state can have functionality as any Java class can.

Execution Preconditions

A clp command can have any number of execution preconditions. See class ClpCmd.Precondition in the API and command Cmd5 in class clp.demo.ConsoleCmds. If a precondition is not satisfied, the clp throws an exception with an error message. clp has one built-in exception, ClpCmd.StateParmRequired, which causes an exception if the specified state parameter does not exist. See the API.

Defining Commands

An application can have any number of application-specific commands, which must be defined programmatically in a specific way. Classes clp.demo.ConsoleCmds and clp.demo.GuiCmds provide step-by-step examples. See also the API. The Clp displays error messages as necessary, such as "Duplicate argument" or "option has no type indicator" (see section "Command Grammar"). Commands can be organized into files in any way. In the clp.demo package, Console commands are in one class and Gui commands in another. Every application-specific command must be added to the Clp; see classes clp.demo.ConsoleProg and clp.demo.GuiProg as examples.

Executing Commands

Commands can be executed via a user interface, Console or Gui, or programmatically (see class clp.Clp in the API). Commands entered at the OS window prompt run on the program thread; commands entered via a Swing component run on the Swing Event Dispatch thread. If an entered command can not be parsed -- does not match any defined command -- the Clp displays message "command not recognized". If there is command execution error, the Clp displays an exception with an error message.

Command Files

Any number of clp commands can be put into a file of commands. A clp command file has extension .clp and contains clp commands and, possibly, comments, and calls to other command files, one per line. A call to another command file consists of the name of the other file. Command files must be in the state parameter $rootfolder folder or a subfolder of it. The $list command lists these command files. The command to execute a command file must be the file's path relative to $rootfolder. For example, if $rootfolder is c:\root and the path to the command file is c:\root\abc\doit.clp ,the command entered would be abc\doit. It could also be abc/doit; clp handles Windows and Linux path delimiters. There is a special command file, clpinit.clp, that can be used to initialize a clp. The file must be in the directory in which the clp is executed (the current directory).

A state parameter can be use to prefix a command file call. Say, for example, there is state parameter `path:abc`. Then command file `abc/doit` could be called using `@path/doit`.

Ending a Program

The `$end:all` command always ends a `clp` program. There may be times, though, when more control is needed for ending the `clp`. For example, say command file `aaa` ends with `$end:this`. Now say command file `bbb` calls `aaa`, then `bbb`. When `aaa` executes it ends the `clp`, so `bbb` does not execute, even if there is no error in `aaa`. The solution is to put a `$noend:push` command at the start of `aaa` and a `$noend:pop` at the end. This prevents the `$end:this` from ending execution. See demo command file `demo1` in the `clp.demo` folder. The `$noend` commands do not affect execution exceptions.

Making a Console Program

To make a `Console clp` program, subclass `clp.AbstractConsoleProg`. Class `clp.demo.ConsoleProg`, demoed in the tutorial video, is an example. The class source code has explanatory comments. `clp.AbstractConsoleProg` has two startup arguments:

- `rootdir`, which is the path to the root directory for command files. All command files must be in that directory or a child of that directory. The default is the user's current direcotory.
- `startfile`, which is the path, relative to `rootdir`, of the command file to execute when the `clp` starts. The default is none.

Making a GUI Program

The easiest way to make a `Gui clp` program is to subclass `clp.AbstractGuiProg` and use class `Clp.DefaultAppPanel`. `clp` as the application panel. Class `clp.demo.GuiProg`, demoed in the tutorial video, is an example. The `clp.demo.GuiProg` source code has explanatory comments. `clp.AbstractGuiProg` uses a button to toggle between two views in the program window:

- View 1: A panel with `JTextArea` command logger at the top and a `JTextField` at the bottom, for command entry.
- View 2: A message panel at the top and a `Clp.DefaultAppPanel` application panel below it. (See the `clp` API.) All `clp` base `Console` and `Gui` commands can be used. `Gui` commands `$buttonrow` and `$button` can be used to put buttons of class `clp.CmdFileButton` into the `Clp.DefaultAppPanel`. The `$buttonrow` and `$button` commands can be entered by the user or be in command files. Each button has an associated `clp` command file. When the user clicks a button, the command file is executed. GUI output is displayed in the `Clp.DefaultAppPanel`. `clp Gui` command `$clickbutton` , entered by the user or with a command file, can be used to click a button programmatically. GUI functions can be executed by a command file just as if the user clicked the buttons.

Custom `GUI` functionality can be obtained by replacing `Clp.DefaultAppPanel` with a subclass of `Clp.AppPanel`, and set it to be used instead of `Clp.DefaultAppPanel`. You could, for example, write a class that uses menus or other Swing components. Using class `clp.CmdFileButton`, you could add command file buttons to the `Clp.AppPanel` subclass. Class `clp.demo.CustomGuiProg` is an example of a custom `Clp.AppPanel`. The `clp.demo.CustomGuiProg` code is commented.

Command Logging

By default a `Clp` sets up a file to which all commands are logged. The file is `clplog.txt` located in the directory in which the `clp` is executed. With a `Console clp`, commands are written to the OS window and to `clplog.txt`. With a `Gui clp` subclassed from `clp.AbstractGuiProg`, commands are written to the logging panel and `clplog.txt`. With base command `$logfile`, logging to `clplog.txt` can be turned on and off. With class `MLogger`, an application can customize the `clp` logging; see the API.