Utilisation de Maven Lite et description de toutes les fonctionnalités

Table des matières

- Utilisation de Maven Lite et description de toutes les fonctionnalités
  - Table des matières
  - Options List
  - Basic Usage
  - Command Line
  - Configuration File
  - Project Creation
  - Project Compilation
  - Project Launch
  - Project Compilation and Launch
  - Project Launch and Compilation
  - Quiet Mode
  - Verbose Mode
  - Exclude Files and Folders
  - Compile to JAR File
  - Launch JAR File
  - Integrate Unit Tests
  - Project Source
  - Compiled Files Destination
  - Resources
  - Classpath
  - Libraries
  - Arguments
    - Example of using the Arguments option
    - Command Line Arguments
      - Command line under Linux and MacOS
      - Command line under Windows
    - Arguments in the Configuration File
  - Main Class
  - Encoding
  - Export
  - Convert to Maven Project
  - Version
  - Help
  - Clear Compiled Files

Options List

- `-f`, `--file`: Load options from a configuration file.
- `-cr`, `--create`: Create the project structure along with a default configuration file.
- `-mvc`, `--model-view-controller`: Specify to the '`--create`' option to create the project structure in MVC style.
- `-c`, `--compilation`: Compile the project.
- `-l`, `--launch`: Launch the project.
- `-cl`, `--compile-launch`: Compile and launch the project (equivalent to -c -l).
- `-lc`, `--launch-compile`: Launch and compile the project (equivalent to -c -l).
- `-q`, `--quiet`: Suppress java output in the terminal during project execution.
- `-v`, `--verbose`: Display executed commands.
- `-ex`, `--exclude`: Exclude java files and folders from compilation.
- `-cj`, `--compile-jar`: Create a jar file of the project.
- `-lj`, `--launch-jar`: Launch an executable jar file.

- `-it`, `--integrate-test`: Integrate unit tests into the project.
- `-s`, `--source`: Folder containing java files to compile.
- `-t`, `--target`: Output folder for compiled files.
- `-r`, `--resources`: Folder containing resource files to be copied to the output folder.
- `-cp`, `--classpath`: Specify the classpath to use during compilation and launch.
- `-lib`, `--libraries`: Folder containing jar files used by the program.
- `-args`, `--arguments`: All arguments to pass to the main class.
- `-m`, `--main`: Main class to launch.
- `-e`, `--encoding`: Change the encoding of java files to compile.
- `-exp`, `--export`: Create an executable jar file to run the project without installing MavenLite.
- `-mvn`, `--maven`: Convert the project to a Maven project.
- `-V`, `--version`: Display the version.
- `-h`, `--help`: Display help and exit.
- `-clr`, `--clear`: Delete files in the output folder.

Basic Usage

- Navigate to your Java project folder and execute the following command:

```
mvnl [options] [arguments]
```

Command Line

The command line is the traditional method to use Maven Lite, although it is less convenient than using a configuration file.

You can include as many options as you want, in any order.

Just separate options with spaces, and put arguments in quotes, for example, `-args "your argument"`. Under Windows, it is not possible to use the `"` character in command line arguments.

- Example of a command line with arguments
    - In this example, we will compile and launch a Java project with the source folder `src/main/java`, the target output folder `target`, the library folder `src/main/resources/lib`, and display executed commands.

```
mvnl --source src/main/java --target target --libraries src/main/resources/lib --
verbose -cl
```

Configuration File

The use of a configuration file is done with the `--file` or `-f` option.

The configuration file is unique for each project and configures the options that Maven Lite should use.

The default name for the configuration file is `LPOM.conf`, and it should be at the project's root. You can rename it, but then you need to specify its name when using the option, for example, `mvnl -f myFile.extension`.

If you want to add your system CLASSPATH to Maven Lite

's CLASSPATH, you can do it in the configuration file, in the `systemClasspath` property.

- Example of a configuration file with options
    - In this example, we will create a configuration file with the project source folder set to `src`, the target output folder set to `out`, and the encoding set to `UTF-8`.

```
mvnl -cr
mvnl --source src --target out --encoding UTF-8
```

Project Creation

The project creation is done with the `--create` or `-cr` option.

The project structure created is a minimal project with a source folder, a target folder, and a configuration file.

You can add more folders and files to your project by adding them in the configuration file.

- Example of project creation
    - In this example, we will create a project with the default configuration.

```
mvnl --create
```

- Example of project creation with MVC structure
    - In this example, we will create a project with the default configuration and the Model-View-Controller (MVC) structure.

```
mvnl --create --model-view-controller
```

Project Compilation

The project compilation is done with the `--compilation` or `-c` option.

You can compile your project without launching it.

- Example of project compilation
    - In this example, we will compile a project with the default configuration.

```
mvnl --compilation
```

Project Launch

The project launch is done with the `--launch` or `-l` option.

You can launch your project without compiling it.

- Example of project launch
    - In this example, we will launch a project with the default configuration.

```
mvnl --launch
```

Project Compilation and Launch

The project compilation and launch can be done with the `--compile-launch` or `-cl` option.

This option is equivalent to using `--compilation` and `--launch` separately.

- Example of project compilation and launch
    - In this example, we will compile and launch a project with the default configuration.

```
mvnl --compile-launch
```

Project Launch and Compilation

The project launch and compilation can be done with the `--launch-compile` or `-lc` option.

This option is equivalent to using `--launch` and `--compilation` separately.

- Example of project launch and compilation
    - In this example, we will launch and compile a project with the default configuration.

```
mvnl --launch-compile
```

Quiet Mode

The quiet mode is done with the `--quiet` or `-q` option.

This option suppresses java output in the terminal during project execution.

- Example of quiet mode
    - In this example, we will compile and launch a project with the default configuration in quiet mode.

```
mvnl -cl --quiet
```

Verbose Mode

The verbose mode is done with the `--verbose` or `-v` option.

This option displays executed commands in the terminal.

- Example of verbose mode
    - In this example, we will compile and launch a project with the default configuration in verbose mode.

```
mvnl -cl --verbose
```

Exclude Files and Folders

Excluding files and folders during compilation is done with the `--exclude` or `-ex` option.

This option allows you to specify files or folders to exclude from the compilation process.

- Example of excluding files and folders
    - In this example, we will compile a project excluding the files in the `excluded` folder.

```
mvnl -c --exclude excluded
```

Compile to JAR File

Compiling your project to a JAR file is done with the `--compile-jar` or `-cj` option.

This option creates a JAR file of your project in the target folder.

- Example of compiling to a JAR file
    - In this example, we will compile a project and create a JAR file in the target folder.

```

```

```
mvnl -cj
```

Launch JAR File

Launching your project from a JAR file is done with the `--launch-jar` or `-lj` option.

This option launches an executable JAR file in the target folder.

- Example of launching from a JAR file
    - In this example, we will launch a project from an executable JAR file in the target folder.

```
mvnl -lj
```

Integrate Unit Tests

Integrating unit tests into your project is done with the `--integrate-test` or `-it` option.

This option adds a testing framework to your project and generates sample test files.

- Example of integrating unit tests
    - In this example, we will integrate unit tests into a project.

```
mvnl --integrate-test
```

Project Source

Setting the project source folder is done with the `--source` or `-s` option.

This option allows you to specify the folder containing your Java files to compile.

- Example of setting the project source folder
    - In this example, we will compile a project with the source folder set to `src`.

```
mvnl -c --source src
```

Compiled Files Destination

Setting the output folder for compiled files is done with the `--target` or `-t` option.

This option allows you to specify the folder where Maven Lite will put the compiled files.

- Example of setting the compiled files destination
    - In this example, we will compile a project and put the compiled files in the `out` folder.

```
mvnl -c --target out
```

Resources

Setting the folder containing resource files is done with the `--resources` or `-r` option.

This option allows you to specify the folder containing resource files to be copied to the output folder.

- Example of setting the resources folder

- In this example, we will compile a project with the resources folder set to `res`.

```
mvnl -c --resources res
```

Classpath

Specifying the classpath to use during compilation and launch is done with the `--classpath` or `-cp` option.

This option allows you to specify the classpath for your project.

- Example of specifying the classpath
  - In this example, we will compile and launch a project with a custom classpath.

```
mvnl -cl --classpath lib/*.jar
```

Libraries

Setting the folder containing jar files used by the program is done with the `--libraries` or `-lib`.

This option allows you to specify the folder containing the JAR files used by the program. All JAR files in the specified folder will be added to the classpath during compilation and execution.

You can create subfolders within the libraries folder to better organize your JAR files, and Maven Lite will take them into account.

- Example of using the Libraries option:
  - In this example, we will compile and run a Java project with the `src/main/resources/lib` folder as the folder containing the JAR files used by the program.

```
mvnl -lib src/main/resources/lib -cl
```

Arguments

This option is used with the `--arguments` or `-args` option.

This option allows you to specify all the arguments to pass to the main class of your project.

These arguments will be passed to the main class in the order they are passed to Maven Lite.

Note: Under Windows, it is impossible to use the `"` character in arguments passed on the command line, so it must be included in the configuration file.

Example of using the Arguments option

- Example of a command line with arguments and a configuration file:
  - In this example, we will compile and run a Java project with arguments using the command line and a configuration file.

```
mvnl -args "argument 1" -f --arguments "argument 3" "argument 4" -args -cl
```

  - Configuration file

```
# Add an argument to the main class
-args "argument 2 (in config)"
```

- Arguments passed to the main class

```
String args = new String[]{"argument 1", "argument 2 (in config)", "argument 3", "argument 4"};
```

Command Line Arguments

Command line under Linux and MacOS

- To pass arguments to the main class of your project using the `-args` and `--arguments` options to avoid any escaping issues.
  - Special characters on the command line are: `\`, `"`
    - `-args '"examp\"le"'` becomes `examp"le`
    - `-args '"-example"'` becomes `-example`
    - `-args '"--example"'` becomes `--example`
    - `-args '"examp\\le"'` becomes `examp\le`
    - `-args '"examp\\\"le"'` becomes `examp\"le`.

Command line under Windows

// TODO: Verify that special characters work under Windows

- To pass arguments to the main class of your project using the `-args` and `--arguments` options to avoid any escaping issues.
  - Special characters on the command line are: `\`
    - `-args '"examp\\\"le"'` becomes `examp\"le`
    - `-args '"-example"'` becomes `-example`
    - `-args '"--example"'` becomes `--example`
    - `-args '"examp\\le"'` becomes `examp\le`
    - `-args '"examp\le"'` becomes `examp\le`
    - `-args '"examp\\\le"'` becomes `examp\\le`
    - `-args '"examp\\\\\"le"'` becomes `examp\"le`
    - `-args '"examp#le"'` becomes `examp#le`
    - `-args '"examp\ le"'` becomes `examp\ le`

Arguments in the Configuration File

// TODO: Verify special characters

- To pass arguments to the main class of your project using the `-args` and `--arguments` options to avoid any escaping issues.
  - Special characters in the configuration file are: `\`, `"`
    - `-args "examp\"le"` becomes `examp"le`
    - `-args "\-example"` becomes `\-example`
    - `-args "\--example"` becomes `\--example`
    - `-args "examp\\le"` becomes `examp\le`
    - `-args "examp\le"` becomes `examp\le`
    - `-args "examp\\\le"` becomes `examp\\le`
    - `-args "examp\\\"le"` becomes `examp\"le`
    - `-args "examp#le"` becomes `examp#le`
    - `-args "examp\ le"` becomes `examp\ le`

Main Class

This option is used with the `--main` or `-m` option.

This option allows you to specify the main class to launch with the package in the form `package.MainClass`. This option is only useful if you have multiple main classes in your project; if not, Maven Lite will automatically find and use the main class of

the project.

- Example of using the Main Class option:
    - In this example, we will compile and run a Java project with the main class `Main2` from the `com.example` package and the default output folder for compiled files.

```
mvnl -m com.example.Main2 -cl
```

Encoding

This option is used with the `--encoding` or `-e` option.

This option allows you to specify the encoding of the Java files to be compiled.

- Example of using the Encoding option:
    - In this example, we will compile and run a Java project with the `ANSI` encoding and the default output folder for compiled files.

```
mvnl -e ANSI -cl
```

Export

This option is used with the `--export` or `-exp` option.

This option allows you to create an executable .class file configured for your project, allowing you to compile and run your project without installing Maven Lite.

The only options accepted by the .class file are the main class options, `--arguments`, `-args`, launch and compile options, `--launch`, `-l`, `--compile`, `-c`, `--compile-launch`, `-cl`, `--launch-compile`, `-lc`, and the configuration file option `--file`, `-f`. Other options will not be taken into account.

It is important to use the configuration file when using the `--export` or `-exp` option as it specifies the parameters that the executable .class file will use.

The purpose of this option is to be able to launch your project via the execution of a single file without arguments and without having to install Maven Lite. This allows people who do not have Maven Lite installed to use your project.

- Example of using the Export option:
    - In this example, we will create an executable .class file configured for our project, allowing us to compile and run our project without installing Maven Lite.

```
mvnl -f -exp
```

    - Configuration file

```
# Source
--source src/main/java

#Output folder for compiled files
--target target

# List of libraries to add to the classpath
--libraries src/main/resources/lib

# Display executed commands
--verbose
```

Convert to Maven Project

This option is used with the `--maven` or `-mvn` option.

This option allows you to convert your project to a Maven project by creating a `pom.xml` file and moving files as needed.

Version

This option is used with the `--version` or `-V` option.

This option displays the version of Maven Lite, the location of the main Maven Lite file, the Java version, the type of build, the Java runtime used, the system's default locale, the platform encoding used by Maven Lite, the operating system name, the operating system version, the operating system kernel version, and the system architecture.

- Example

```
Maven Lite 2.0.0
Maven Lite home: /usr/local/etc/maven-lite/
Java version: 17.0.9, vendor: Private Build, runtime: OpenJDK Runtime Environment
Default locale: fr_FR, platform encoding: UTF-8
OS name: "Linux", version: "6.5.0-14-generic", architecture: "amd64"
```

Help

This option is used with the `--help` or `-h` option.

This option displays the list of options along with their description, the number of arguments they take, their default value if they have one, and a link to the documentation.

Clear Compiled Files

This option is used with the `--clear` or `-clr` option.

This option deletes all files in the output folder for compiled files. This option frees up disk space and provides a clean JAR file.