muScript User Manual

1 Introduction

muScript (μScript) is a command line version of the muJava (μJava) mutation analysis tool. It uses μJava's core mutation engine. The purpose of μScript is to facilitate experimental studies using mutation testing. By combining a few flexible commands, users can design scripts, run a large amount of tests, and collect execution results without using GUI interface of μJava.

As with μJava, μScript is provided on an "as is" basis. We have no funds and very limited resources to provide support or even respond to queries.

μScript has four commands, as listed in Table 1. They are all included in the μJava .jar file and called with the syntax "`java mujava.cli.command`."

Table 1: μScript Commands

Command	**Description**	**Section**
testnew	Create a test session	3
genmutes	Generate mutants	4
runmutes	Run test cases against mutants	6
markequiv	Mark mutants as equivalent	7

2 Setting up the μScript Configuration File

Before running μScript, it is necessary to set the correct directory location and other configuration information. The configuration file is named *mujava.config*. It is an extended version of the μJava configuration file, with the possibility of adding additional options.

1. Home directory location: *MuJava_HOME* =
   An absolute path is required so that μScript can find classes under test, find test sets, to save results. For example, in Windows, we can set it as *"c:\mujava"*; in a Mac, Linux, or Unix machine, it could be *"/Users/username/mujava"*

2. Debug mode: (optional) *Debug_mode* =
   We can set the *debug_mode* to either "true" or "false." Enabling *debug_mode* will display more detailed intermediate results on the console.

3. Classpath:
   To get μScript running, six library jar files must be defined correctly in the classpath:
   mujava.jar, *commons-io.jar*, *openjava.jar*, *tools.jar*, *junit.jar*, and *hamcrest-core.jar.*

For example, on a Windows platform, if all the jar files are stored in the directory *“c:\mujava”*, extend *“CLASSPATH”* in the *“Environment Variables”* by adding the paths:

```
c:\mujava\junit.jar;c:\mujava\hamcrest-core-1.3.jar;c:\mujava\mujava.jar;
c:\mujava\openjava.jar;c:\mujava\commons-io-2.4.jar
```

on a Unix/Linux platform, if all the jar files are stored in the directory *“/home/ldeng2/mujava”*, the following command will define them in the classpath:

```
export CLASSPATH=‘‘$CLASSPATH:/home/ldeng2/mujava/mujava.jar:
/home/ldeng2/mujava/commons-io-2.4.jar:/home/ldeng2/mujava/openjava.jar:
/usr/local/java/jdk1.7.0/lib/tools.jar:/home/ldeng2/mujava/junit.jar:
/home/ldeng2/mujava/hamcrest-core.jar’’
```

3 Creating a Test Session

Command: *testnew*

A *test session* defines an independent mutation test on a program component. It comprises source files (in Java), bytecode class files, JUnit tests, and results. These are stored in four directories that *testnew* creates, *testset*, *result*, *classes*, and *src*.

Figure 1 shows the directories created by *testnew* for the session *session1*.

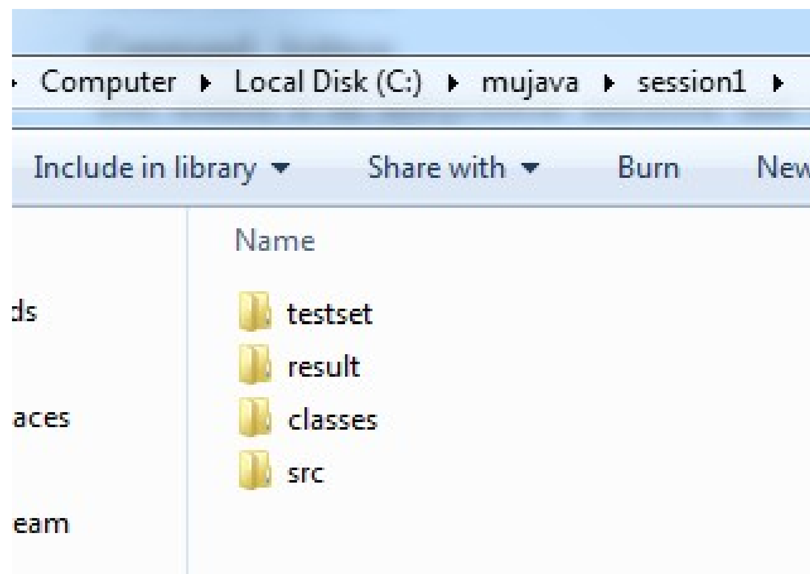

Figure 1: Test Session Directory

The program *testnew* also prepares for subsequent processes, such as compiling Java source files and copying compiled class files into the new session created.

Parameters:

1. <session name> <src file name 1> <src file name 2> ... <src file name n>}

The "`<session name>`" will be used as the name of the new session and the root directory of the test session. If a session with that name already exists in the home directory of μScript, an error message will be returned and the program will exit without making any changes. The source file names can be given with or without the *.java* extension. Normally, one test session contains only one class under test, however, it is possible to put multiple classes in one session. This is normally done when the class under test uses another class.

Figure 2 shows four example commands that can be used to create a new test session for the Java source code named *cal.java* saved in *c:\mujava\src* folder.

```
C:\mujava>java mujava.cli.testnew session1 cal
Source file is compiled successfully.
Session is built successfully.

C:\mujava>java mujava.cli.testnew session2 cal.java
Source file is compiled successfully.
Session is built successfully.

C:\mujava>java mujava.cli.testnew session3 c:\mujava\src\cal
Source file is compiled successfully.
Session is built successfully.

C:\mujava>java mujava.cli.testnew session4 c:\mujava\src\cal.java
Source file is compiled successfully.
Session is built successfully.
```

Figure 2: Example Commands for Creating New Test Sessions

Java source files should usually be saved in the *src* folder of the home directory, so that only file names are required when creating a new session (the first example command in Figure 2).

However, μScript is also able to fetch source files from other directories with an absolute path, such as shown in the last two commands in Figure 2.

2. *<-debug>*

   If the configuration file does not set the *debug* mode, it can be set as an option to *testnew*. Options given on the command line have higher priority than the configuration file, so they override the settings in the file. Note the *debug* option is available with ALL subsequent commands. An example is shown in Figure 3.

4 Generating Mutants

Command: *genmutes*[1]

[1] Prerequisite: Make sure the current version of Openjava.jar from the μJava website is correctly set in the system's environment.

```
C:\mujava>java mujava.cli.testnew -debug session1 cal

Make C:\mujava\session1 directory...
Making C:\mujava\session1 directory  ...done.

Make C:\mujava\session1\src directory...
Making C:\mujava\session1\src directory  ...done.

Make C:\mujava\session1\classes directory...
Making C:\mujava\session1\classes directory  ...done.

Make C:\mujava\session1\result directory...
Making C:\mujava\session1\result directory  ...done.

Make C:\mujava\session1\testset directory...
Making C:\mujava\session1\testset directory  ...done.
Source file is compiled successfully.
Session is built successfully.
```

Figure 3: Creating a New Test Session With the *Debug* Mode

```
C:\mujava>java mujava.cli.genmutes -ROR -AOIS session1
1 : cal.java
File C:\mujava\session1\src\cal.java
----------------------------------------------------------
Total mutants gnerated for cal.java: 104
```

Figure 4: Generating ROR and AOIS Mutants

The program *genmutes* is used to generate mutants. The current version is only able to generate method level mutants.

Parameters:

1. -<operator name 1> -<operator name 2> ... <session name>

   Each "<operator name>" is one of {AORB, AORS, AOIU, AOIS, AODU, AODS, ROR, COR, COD, COI, SOR, LOR, LOI, LOD, ASRS, ALL}. The operator names can be either lower or upper case. If no "-<operator name>" is provided, by default, μScript will generate mutants for all operators. This is equivalent to using the name "*ALL*."

   The "<session name>" takes the name of a test session that already exists.

   For example, Figure 4 shows a command to generate all ROR and AOIS mutants for the class in *session1*. Note that μJava discards mutants that do not compile. μScript prints the total number of mutants generated.

   As shown in Figure 5, each mutant has its own folder in which it saves all necessary information. All mutants generated will be stored in the path:
   muScriptHome/sessionName/result/Classname/traditional_mutants

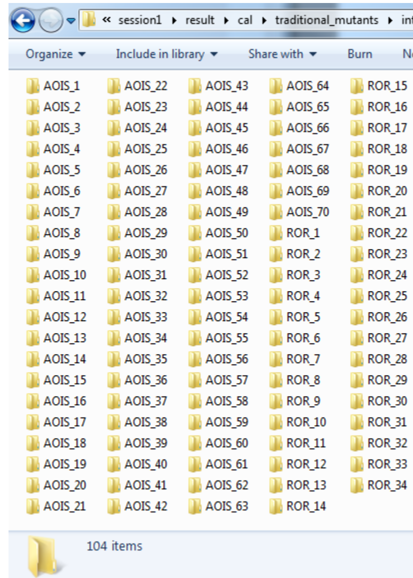

Figure 5: ROR and AOIS Mutants in the *result* Folder

2. *−all <session name>*

   The "*−all*" option enables all 15 traditional method-level mutation operators, as shown in Figure 6. Thus, the *cal.java* class has 173 mutants. Also, if the current session has more than one class, *genmutes* will generate mutants for all classes.

```
C:\mujava>java mujava.cli.genmutes -ALL session1
1 : cal.java
File C:\mujava\session1\src\cal.java
--------------------------------------------------
Total mutants gnerated for cal.java: 173
```

Figure 6: Generating ALL mutants

5 Creating Tests

μScript uses JUnit tests that are supplied by the tester. They can be created in any IDE or command line tools, and must be compiled to *.class* files outside of μScript. As shown in Figure 7, the *.class* files need to be put in the folder *session_name/testset/*, which is created by the *testnew* command. The tester can add new JUnit test files at any time.

Thus, a common process might be to create a few tests, run the mutants on the tests, then examine the live mutants and either mark them as equivalent or generate additional tests. This process can repeat until the tester is satisfied with the mutation score. In this way, the mutants become "guides" to help the tester design good tests.

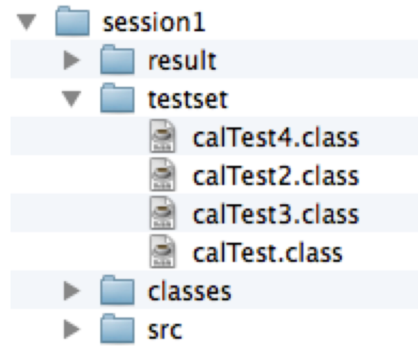

Figure 7: JUnit Test Files in the *testset* Folder

6 Running Tests

Command: *runmutes*[2]

The program *runmutes* runs tests against mutants. It saves results in two files in the result folder, named *result_list.csv* and *mutant_list*. The result files are used in additional calls to *runmutes*, and by the *markequiv* command. *result_list.csv* can be opened in Excel or other spreadsheet applications, and *mutant_list* can be opened in any text editor.

runmutes can run in one of three modes:

1. *(-default)* Each test is run against mutants that are alive (not killed or marked equivalent). This is the typical way mutation testing is used in practice.

2. *(-dead)* Each test is run against all live and dead mutants (but not equivalent mutants). This supports experimental studies. An additional option *-equiv*, can be added to run the tests against all equivalent mutants as well.

3. *(-fresh)* A new results file with a time stamp is generated. The old results file is not used, thus all mutants start as alive, and all tests are run against all live mutants.

The *-default*, *-dead*, and *-fresh* modes are mutually exclusive.

Parameters:

1. `-<operator name1> -<operator name2> ... <testset name> <session name>`

   The “`<operator name>`” is one of {AORB, AORS, AOIU, AOIS, AODU, AODS, ROR, COR, COD, COI, SOR, LOR, LOI, LOD, ASRS, ALL}. The operator names can be either lower or upper case. If no operator name is provided, by default, μScript will run tests against all mutants. This is equivalent to using the name “*ALL*.”

   The “`<session name>`” is an already existing test session.

[2] Tip: Make sure JUnit is correctly configured in the system environment.

The "<testset name>" is the name of a JUnit test file, without the *.class* extension. If no testset name is given, μScript will run all test sets in the *testset* directory.

Figure 8 shows the test set *calTest* being executed against all mutants in the session *session1*. After execution, the total number of mutants killed, the number of total mutants, and the mutation score are displayed.

```
C:\mujava>java mujava.cli.runmutes calTest session1
Class Name: cal
Test Name: calTest
-----------------------------------------------
Running
Current running mode: default
........................
Total mutants killed: 80
Total mutants: 104
Mutation Score: 0.7692
Please look at the result files (mutant_list and result_list.csv) for details.
Use "markequiv" command to mark equivalent mutants.
```

Figure 8: Running Mutants in *default* Mode

2. `-all <testset name> <target session name>`

   The "`-all`" option runs the test set against all 15 of the traditional method-level mutation operators.

3. `-p <random percentage>`

   Sometimes we would like to randomly choose a subset of the total mutants, so the "`-p`" option is provided. This is a common experimental technique. We can specify a number from 0 to 1 as the random percentage. For example Figure 9 uses "`-p 0.5`" to run tests against 50% of the live mutants, chosen randomly.

```
C:\mujava>java mujava.cli.runmutes -p 0.5 calTest session1
Class Name: cal
Test Name: calTest
-----------------------------------------------
Running
Current running mode: default
....................................................
Total mutants killed: 42
Total mutants: 52
Mutation Score: 0.8077
Please look at the result files (mutant_list and result_list.csv) for details.
Use "markequiv" command to mark equivalent mutants.
```

Figure 9: Running Tests Against 50% of the Mutants, Randomly Chosen

7 Marking Mutants as Equivalent

Command: *markequiv*

The program "*markequiv*" is used to mark live mutants as equivalent.

Parameters:

1. `<class name> <mutant 1> <mutant 2> ... <mutant n> <session name>`

   The "`<class name>`" specifies the name of the target class. The options "`<mutant 1> <mutant 2> .` specifies names of live mutants, as listed in *result_list.csv*. The "`{<session name>`" takes the session name.

   The example in Figure 10 marks the mutant named *ROR_8* of class *cal* in *session1* as equivalent. If a mutant is already dead, μScript will skip it; if a mutant does not exist, it will be ignored.

```
C:\mujava>java mujava.cli.markequiv cal ROR_8 session1
All equivalent mutants are marked.
```

Figure 10: Marking an Equivalent Mutant