Using Javagui's Test Modes

Thomas Behr

2006-04-18

1 The Simple Test mode

In the simple test mode, no user interaction is required in performing a script given at start. Javagui is started in this mode using the command:

```
sgui --testmode [<filename>]
```

The effect of the option `--testmode` is to disable all message windows. Indeed all messages are written to the console. If a filename is given, the content of this file will be executed in the same way as when the user will input the commands contained in this file in the command panel. In particular, this is the same as entering a name for the start script in `gui.cfg` except the user must not confirm any messages, e.g. license or the category from the HoeseViewer.

1.1 File Format for Simple Tests

The format of the test files in this mode depends on the setting of the `SCRIPT_STYLE` variable in the `gui.cfg` file. This variable can hold the values `gui` or `tty`. If it set to `gui`, each command must be written in a single line. Multi-line commands are not possible. If the value of `SCRIPT_STYLE` is `tty`, the testfile has to be in the same format as a TTY script. In particular, each command is finished by an empty line or a ';' at the end of a line. Using the @ directive, you can 'import' files for execution. Note that the filenames must be given relative to the start directory of Javagui or absolute, not relative to the location of the main script file.

2 The Extended Testmode

Start the extended testmode by:

```
sgui --testmode2 <filename>
```

where <`filename`> is the name of the test file. In contrast to the simple test, the filename is required. If this option is used, Javagui will start and execute the tests specified in the test file. After the tests, Javagui will stay about ten seconds on the screen and exit automatically. As in the simple testmode, all messages are written to the console instead into message windows. The return code of Javagui will be the number of errors occurred. The format of the test file will be described in the next section.

2.1 File Format for the Extended Test

The test file for extended tests is in textual nested list format. The rough structure is:

```
( <setup> <tests> <teardown> )
```

with:

```
<setup>     ::=  ( <command_1> ...  <command_n> )
<tests>     ::=  ( <test_1> ...  <test_m> )
<teardown>  ::=  ( <command_1> ...  <command_k> )
```

A command may be a string atom, a text atom, or a list consisting of a sequence of text atoms and string atoms. Formally:

```
command ::= <string atom> |
            <text atom> |
            ( {<text atom> | <string atom> }* )
```

While the first two formats enable easy definition of a command, the last format allows one to define complex commands containing text atoms. An example is:

```
( 'restore database germany from '
  "'" '../../secondo-data/Databases/germany' "'" )
```

All atoms are connected without any additional spaces.

Each test consists of the command and the result:
`test ::= ( <command> <result>)`
The command is formatted as in the other sections.
A result consists of a list with two elements:

```
<result> ::= ( <format> <resultspec> )
```

The format is a symbol value holding the value `file` or `list`.

```
<format> ::= file | list
```

The result specification is defined as:

```
<resultspec> ::= () |
                 <success>  |
                 ( <success> <expresult> ) |
                 ( <success> <expresult> <epsilon> )
```

If the result specification is an empty list, no tests are performed, only the command will be executed. <success> is a boolean atom describing the success of a query.

```
<success> ::= TRUE | FALSE
```

If it is the only result specification or the value is false, the computed result list is not compared with the expected result list.
The required format of the expected result <`expresult`> depends on the value of the result format. If the format is specified as `list`, `expresult` can be any list. This list describes the result of the command. If the format is given as `file`, `expresult` has to be a string atom or a text atom. The actual expected result is read from a file with name specified in this atom. If the file does not exist, the test fails. <`epsilon`> is a real atom holding the maximum allowed deviation between real value in comparisons of nested lists. If epsilon is omitted, zero is assumed which leads to an exact comparison.

3 The TestRunner Mode

In this mode, Javagui processes files in the syntax used in the TestRunner of secondo. In contrast to the normal testrunner, sql-like commands for the optimizer are possible. The format is described in the documentation of the TestRunner. Javagui must called via the following command to run in the mode:

```
sgui --testrunner <testfile>
```

After the execution of the testfile, Javagui waits for about 5 seconds and exists.