Running FrAid

NOTICE: ""

Table of contents

1. Introduction

When used as a standalone system, FrAid can process input and generate output from/to variety of sources/destinations. The interpreter works with standard Java streams so the sources or destinations may be disk files, pipes, user input from a console, strings passed from a Java application or even sockets, and this could be done in any combination or sequence (redirect the streams at runtime, read from file, print to the console or vice versa, etc.). Here are some examples (the input/output shown is from a Unix system but pretty much the same applies to Windows; the '$' symbol is the particular system's prompt):

2. From the command line

2.1. Start a FrAid session

Start a FrAid session using the system's console for IO (introductory info is printed, FrAid's prompt appears and processing of user's 2+2; is shown):

```
                    $ java -cp fraid.jar org.fraid.interpreter.Fraid
                    FrAid version 1.0, Copyright (C) 2003 Ivaylo I. Iliev
                    FrAid comes with ABSOLUTELY NO WARRANTY;
                    for details see the GNU General Public License.
                    This is free software, and you are welcome to
redistribute it under the conditions of the license.
                    fraid:1>2+2;                    //user input
                    (4.0 + 0.0i)                    //system's output
                    fraid:2>exit;                   //next user command
                    $
```

2.2. Input from file

Take input from a file named `myFraidProgram` which in this oversimplified example contains a single line:

```
                    2+2;
```

and exit after processing:

```
                    $ java -cp fraid.jar org.fraid.interpreter.Fraid -quiet
-in myFraidProgram
                    (4.0 + 0.0i)
                    $
```

2.3. Input directly from the command line

Take input directly from the command line:

```
                    $ java -cp fraid.jar org.fraid.interpreter.Fraid -in
"2+2;" -quiet
                    (4.0 + 0.0i)
                    $
```

2.4. Using pipes

Use pipe to redirect input (note the extra prompt which appears):

```
                    $ echo "2+2;" | java -cp fraid.jar
org.fraid.interpreter.Fraid -quiet
                    fraid:1>(4.0 + 0.0i)
                    $
```

2.5. Shebang

In Unix's shebang style, here we have a file named `shebang.frd` which contains:

```
                    //bin/true && exec /my/java/location/java -cp
/my/fraid/location/fraid.jar org.fraid.interpreter.Fraid -quiet -in
/my/shebang/location/shebang.frd
                    2+2;
```

It can be excuted either like this:

```
                    $ java -cp fraid.jar org.fraid.interpreter.Fraid -quiet
-in shebang.frd
                    (4.0 + 0.0i)
                    $
```

or like this (after making `shebang.frd` executable):

```
                    $ ./shebang.frd
                    (4.0 + 0.0i)
                    $
```

which allows the file to be executed from anywhere in the system.

3. FrAid's command line

3.1. Synopsis

```
                    java -cp fraid.jar org.fraid.interpreter.Fraid
                    [ -in   system | fraid | line   | box  | FILENAME |
STRING; ]
                    [ -out  system | fraid | dialog | null | FILENAME ]
                    [ -err  system | fraid | dialog | null | FILENAME ]
                    [ -info system | fraid | dialog | null | FILENAME ]
                    [ -logfile               dialog | null | FILENAME ]
                    ( -log      in | out   | err    | info )*
                    ( -syms  CLASSNAME )*
                    ( -symsa CLASSNAME )*
                    ( -load  FILENAME )*
                    [ -debug ]
                    [ -quiet ]
```

Note:

Depending on the operating system, `java -cp fraid.jar org.fraid.interpreter.Fraid` is usually wrapped in `fraid.sh` or `fraid.bat` but you can pass the above commandline arguments as well.

3.2. Options

3.2.1. -in

-in - controls the input source

`system` - takes input from `System.in` , this is the default; will exit when the `exit()` function is called or ^C is pressed;

`fraid` - uses `FraidConsole` for input, automatically redirects the output there as well; will exit when the `exit()` function is called or the console is closed;

`line` - opens a simple dialog with a single-line text area (has command history (Up, Down keys) and parenthesis matching); will exit when the `exit()` function is called;

`box` - opens a simple dialog with a multi-line text area (has command history (Alt-Up, Alt-Down keys) and parenthesis matching); will exit when the `exit()` function is called;

`STRING` - if a string that contains the ';' character is passed it is treated as a FrAid script: it gets executed, prints a result and the system exits; will exit when the whole script is processed and all GraphicsPanels are closed, or before that if the script contains a call to the `exit()` function;

`FILENAME` - takes input from the specified file; will exit when the end of the file is reached, or before that if the script contains a call to the `exit()` function;

3.2.2. -out

-out - controls the output stream

`system` - prints on `System.out` , this is the default;

`fraid` - uses FraidConsole for output, automatically redirects the input there as well;

`dialog` - opens a simple dialog with a text area;

`null` - discards the output;

`FILENAME` - saves the output to the specified file;

3.2.3. -err

-err - controls the error stream

`system` - prints on `System.out` , this is the default;

`fraid` - uses FraidConsole for output, automatically redirects the input there as well;

`dialog` - opens a simple dialog with a text area; if you close the dialog it will reappear with the next error;

`null` - discards the output;

`FILENAME` - saves the output to the specified file;

3.2.4. -info

`system` - prints on `System.out` , this is the default;

`fraid` - uses the bottom TextArea of FraidConsole for output; if the FraidConsole is not used opens a dialog;

`dialog` - opens a simple dialog with a text area;

`null` - discards the output;

`FILENAME` - saves the output to the specified file;

3.2.5. -log

-log - an item in a list of the streams being logged

`in` - the input is logged;

`out` - the output is logged; logged by default;

`err` - the error stream is logged; logged by default;

`info` - the info stream is logged;

3.2.6. -logfile

-logfile - controls the logging destination; the default destination is the file `Fraid/fraid.log` ;

dialog - opens a simple dialog with a text area;

null - doesn't log anything even if there are streams are attached to it;

FILENAME - logs to the specified file (in the `Fraid` directory);

3.2.7. -syms

```
-syms absolute_path/file_name
```

A name of an XML file containing a serialized instance of something implementing the `StaticSymbolsInterface` . The default location is `org/fraid/symbtable/static_symbols` within the jar. With this option this default location will be ignored, so you can select the precise functions that you want loaded.

3.2.8. -symsa

```
-symsa /file_name
```

Same as `-syms` but the default symbols will be loaded.

3.2.9. -load

```
-load FILENAME
```

Loads an item from a list of FrAid scripts to be loaded before any user input is processed. Convenient for loading function definitions before using them.

3.2.10. -debug

Turns debugging on (you can turn it off later with the `debug()` function). When debugging is on, the complete stack trace of the exception is printed;

3.2.11. -quiet

Prevents the printing of the introductory info.

4. Examples

4.1. Example 1

Loading definitions from a file and calling a custom function (based on those definitions). Say we have a file named `lorenz.frd` :

```
clear;
a=10;b=28;c=8/3;
controlVar(a,1,b,1,c,1);

lor1( x1, x2, x3, t ) = a * (x2 - x1);
lor2( x1, x2, x3, t ) = b*x1 - x2 - x1*x3;
lor3( x1, x2, x3, t ) = x1*x2 - c*x3;

x=0; y=1; z=0;
controlVar(x,.1,y,.1,z,.1);

startP=0; endP=100; numberSamples=10000;

rk1( lor1,//the system
lor2,
lor3,
x, y, z, //the initial condition
startP, //the start point
endP, //the end point
numberSamples, /*number of samples*/
"_rk");
```

If we run:

```
$ java -cp fraid.jar org.fraid.interpreter.Fraid -quiet
-load lorenz.frd -in "plot2(_rk_0,_rk_2,0,100);" -out null -info null &
$
```

we get:

Note that FrAid is invisible, only a window with our plot opens (now you can save the picture to file or do whatever you want). The process is run in the background so even the system's prompt appears immediately. Here I've used the picture from the zooming example, but the result is pretty much the same.

With the command above we are telling FrAid the following (the order is unimportant):

- do not print the intro,
- load the file `lorenz.frd` ,
- execute `plot2(_rk_0,_rk_2,0,100)` ,
- ignore output and info streams.

4.2. Example 2

Of course, we can do the same with a function like `lorenz1.frd` (the only difference is the last line):

```
clear;
a=10;b=28;c=8/3;
controlVar(a,1,b,1,c,1);

lor1( x1, x2, x3, t ) = a * (x2 - x1);
lor2( x1, x2, x3, t ) = b*x1 - x2 - x1*x3;
lor3( x1, x2, x3, t ) = x1*x2 - c*x3;

x=0; y=1; z=0;
controlVar(x,.1,y,.1,z,.1);

startP=0; endP=100; numberSamples=10000;

rk1( lor1,//the system
lor2,
lor3,
x, y, z, //the initial condition
startP, //the start point
endP, //the end point
numberSamples, /*number of samples*/
"_rk");

plot2(_rk_0,_rk_1,0,100);
```

and then call:

```
$ java -cp fraid.jar org.fraid.interpreter.Fraid -quiet
-in lorenz1.frd -out null -info null &
```

However, the point of Example 1 was to illustrate that one can load some definitions and use them later. What we asked it to do with those definitions could have been completely different.

""

""