Jekejeke Runtime Swing

Version 1.0.6, May 28th, 2015

XLOG Technologies GmbH

Jekejeke Prolog

Runtime Library 1.0.6

User Manual Swing

Author: XLOG Technologies GmbH
Jan Burse
Freischützgasse 14
8004 Zürich
Switzerland

Date: May 28th, 2015
Version: 0.28

Participants: None

Warranty & Liability

To the extent permitted by applicable law and unless explicitly otherwise agreed upon, XLOG Technologies GmbH makes no warranties regarding the provided information. XLOG Technologies GmbH assumes no liability that any problems might be solved with the information provided by XLOG Technologies GmbH.

Rights & License

> … Defined predicates with arity>0, both static and dynamic, are indexed on the functor of their first argument [1, p.17] ...
>
> [1] Language Reference, Jekejeke Prolog 0.8.1, XLOG Technologies GmbH, Switzerland, February 22nd, 2010

Trademarks

Table of Contents

Change History

Jan Burse, November 9th, 2009, 0.1:

- Initial Version.

Jan Burse, February 26th, 2010, 0.2:

- Added session and debugging.

Jan Burse, April 4th, 2010, 0.3:

- Added consult and new menu items.

Jan Burse, June 2nd, 2010, 0.4:

- Better error handling and settings dialog.

Jan Burse, July 2nd, 2010, 0.5:

- Code styling introduced and first two tours written.

Jan Burse, July 25th, 2010, 0.6:

- License exception handling updated.

Jan Burse, October 2nd, 2010, 0.7:

- Terminal functions enhanced.

Jan Burse, December 29th, 2010, 0.8:

- License exception handling updated and threads tour completed.

Jan Burse, January 2nd, 2011, 0.9:

- Memory low tour completed.

Jan Burse, April 8th, 2011, 0.10:

- Example adapted and interactions enhanced.

Jan Burse, April 15th, 2011, 0.11:

- Register and about dialog enhanced.

Jan Burse, April 23th, 2011, 0.12:

- Terminal interactions moved into separate document.

Jan Burse, Mai 6th, 2011, 0.13:

- Enlists settings and email activation section introduced.

Jan Burse, June 6th, 2011, 0.14:

- Class path example updated.

Jan Burse, August 27th, 2011, 0.15:

- Terminal settings section updated.

Jan Burse, October 1st, 2011, 0.16:

- Colours settings section introduced.

Jan Burse, December 2nd, 2011, 0.17:

- Conversation section moved from language reference.

Jan Burse, December 16th, 2011, 0.18:

- Call-site and ancestor inspection section introduced.

Jan Burse, January 1st, 2012, 0.19:

- Text settings introduced.

Jan Burse, February 14st, 2012, 0.20:

- Conversation removed and tab host console introduced.

Jan Burse, July 19th, 2012, 0.21:

- Better menu accelerators.

Jan Burse, August 8th, 2012, 0.22:

- Document renamed to runtime library and debugger functionality removed.

Jan Burse, September 6th, 2012, 0.23:

- Capability icons introduced.

Jan Burse, December 20th, 2012, 0.24:

- Information panel introduced.

Jan Burse, November 8th, 2013, 0.25:

- Newly detected paths and capabilities dialog introduced.

Jan Burse, March 7th, 2014, 0.26:

- Build menu introduced.

Jan Burse, April 5th, 2014, 0.27:

- New add path menu item introduced and then removed again.

Jan Burse, May 28th, 2015, 0.28:

- New language settings panel introduced.

1 Introduction

The user interface for the Jekejeke Prolog runtime library can be either run with a GUI or without a GUI. In the following we describe the GUI of the Jekejeke Prolog runtime library. It only allows query answering and source consulting. Further capabilities and their activations can be managed.

- **Tours:** Each guided tour shows how a task can be solved via the user interface of the Jekejeke Prolog runtime library. Among the tasks we find Java foreign predicate execution, multiple thread execution and memory low excess.

- **Menus:** The Jekejeke Prolog runtime library does not provide the editing of programs. For this purpose arbitrary external editors can be used. Even editors that come with external integrated development environment are suitable. Nevertheless some action items to control the console and the interpreter are needed come as menus.

- **Toolbars and Popup Menus:** Toolbars can be attached to windows and allow the invocation of action items quicker than menus. Therefore we also provide access to some action items via the toolbar. Further improvements in usability are pop-up menus that can be invoked by right-click mouse.

- **Console Windows & Dialogs:** The console consists of a terminal window for the interaction with a Jekejeke Prolog thread. Besides a text dialog based interaction it is also possible to interact via the menus, the toolbar and the pop up menu with the Jekejeke Prolog thread.

- **Activation Dialogs & Panels:** The activation dialogs & panels deal with the management of the capability store. Only enlisted capabilities are shown. It is possible to interactively activate capabilities.

- **Settings Panels:** The settings dialog shows tabbed panels. Each panel is responsible for a particular set of settings. All the settings are stored in the user profile. Some changes are only effective after a restart of the runtime library.

- **Appendix Tour Listings:** The full source code of the Java classes and the Prolog texts for the tours is given.

2 Tours

Each guided tour shows how a task can be solved via the user interface of the Jekejeke Prolog runtime library. Among the tasks we find Java foreign predicate execution, multiple thread execution and memory low excess.

- **Text Tour:** This guide explains the basic usage of your computing environment and the runtime library. We will step by step show how a Prolog text can be created and executed that will display “Hello World!”.

- **Class Path Tour:** In this guide we want to show how the runtime library can be connected with Java code. We will step by step show how a Java foreign predicate can be created and executed that will hello an argument.

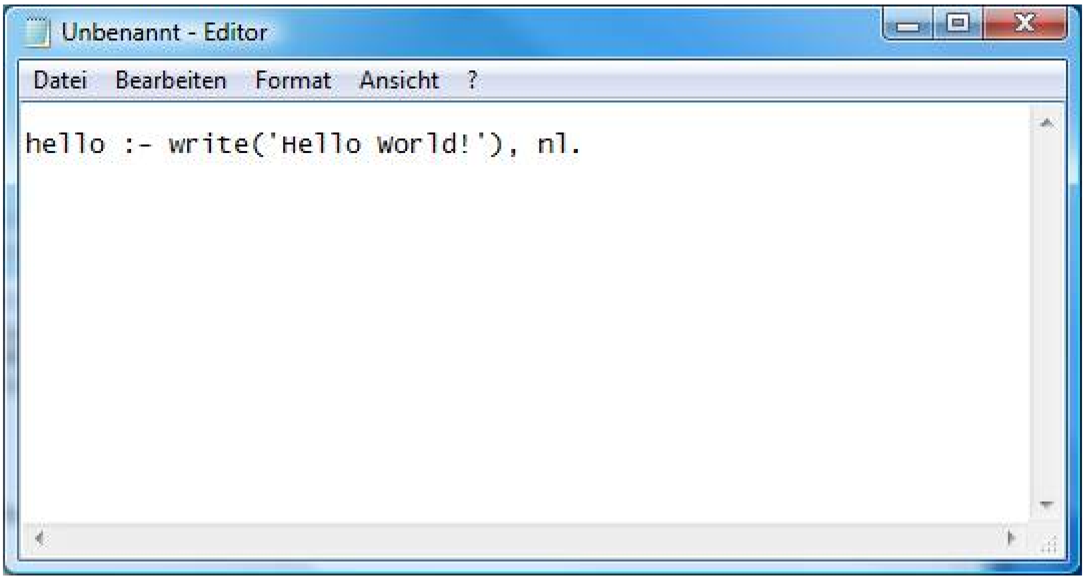
- **Threads Tour:** In this guide we want to show how it is possible to execute easily multiple threads in the runtime library.

- **Memory Low Tour:** The runtime library uses the Java memory manager to check whether the used memory exceeds a threshold.

2.1 Text Tour

This guide explains the basic usage of your computing environment and the runtime library. We will step by step show how a Prolog text can be created and executed that will display “Hello World!”. The current version of the Jekejeke Prolog runtime library does not provide an editor for Prolog texts. Instead you have to use an editor of your choice from your computing environment. A simple text editor will do.

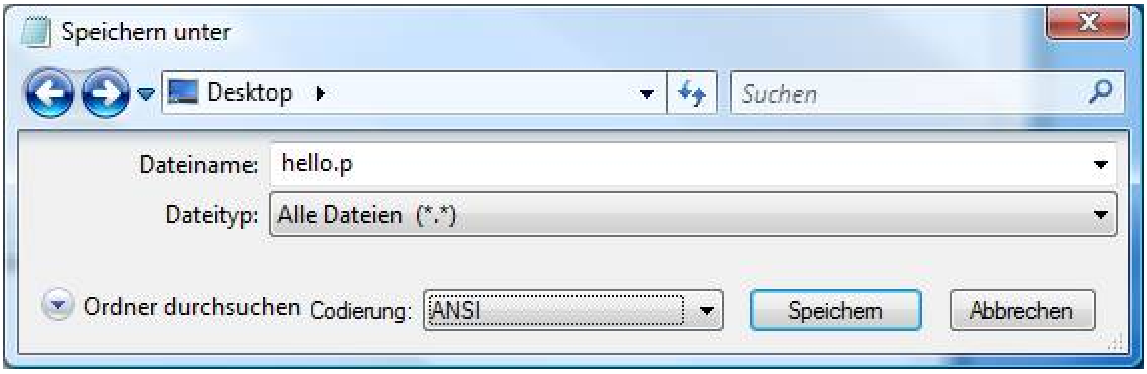
A Prolog text contains facts, rules and directives. Our example will only contain one rule for a predicate hello/0 which will display the text “Hello World!”. The text will be represented by an atom enclosed in single quotes (‘):



```
hello :- write('Hello World!'), nl.
```

Picture 1: Creating a Prolog Text

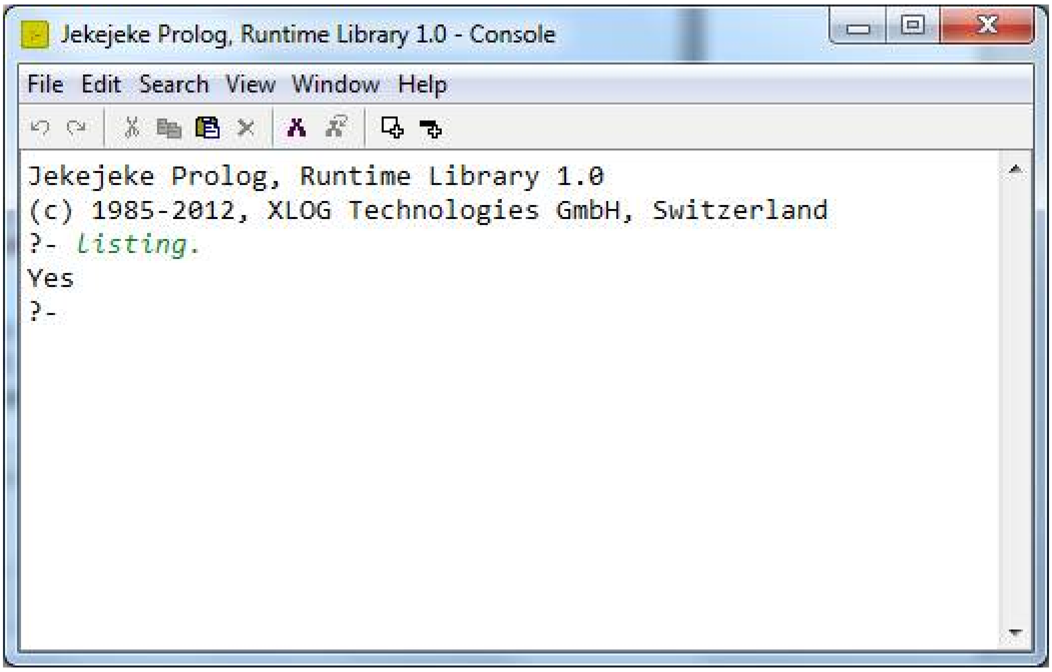
When the Prolog text has been entered you usually will be able to save it to your computing environment. By use of the extension “.p” in the chosen file name you can indicate that the file is a Prolog source. The Jekejeke Prolog interpreter assumes as a default encoding UTF-8 for Prolog texts. There is not yet an encoding option when consulting from files. Therefore it is recommended to store Prolog texts either in UTF-8 or when possible in 7-bit ASCII. 7-bit ASCII is also possible since it is a subset of UTF-8:



Dateiname: hello.p
Dateityp: Alle Dateien (*.*)
Codierung: ANSI

Picture 2: Prolog Text Name and Encoding

You can now start-up the Jekejeke Prolog runtime library. It will show the console window. For more details on starting the runtime library see the installation guide. A quick check with the listing/0 system predicate reveals that the current knowledge base does not yet contain any user predicates:


Jekejeke Prolog, Runtime Library 1.0 - Console
File Edit Search View Window Help

```
Jekejeke Prolog, Runtime Library 1.0
(c) 1985-2012, XLOG Technologies GmbH, Switzerland
?- listing.
Yes
?-
```

Picture 3: The Initial Main Window

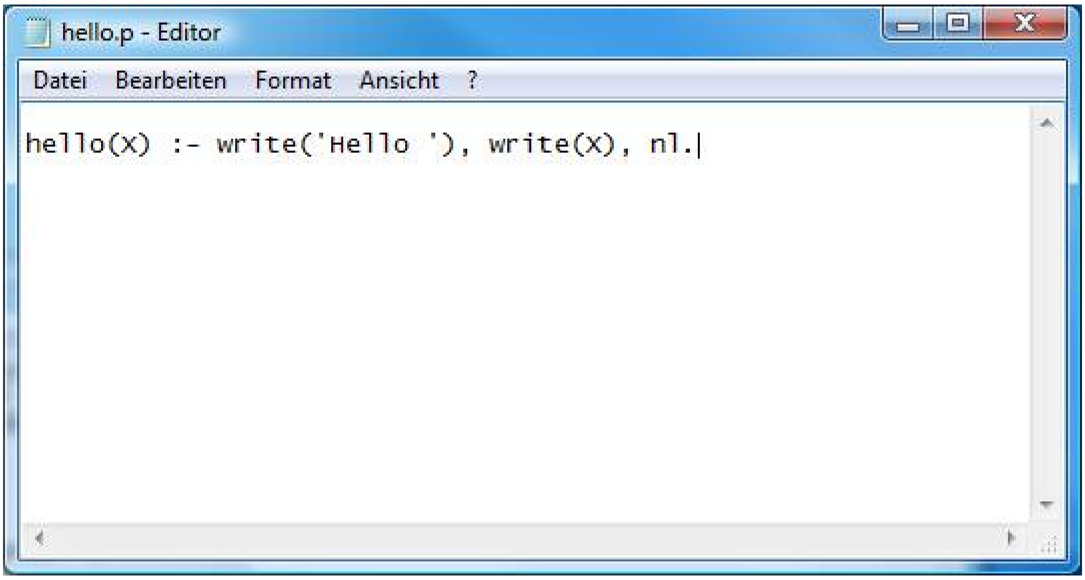
The system predicate consult/1 is suited to read in a Prolog text. As an argument we can use the source path of the Prolog text. A check with the listing/0 system predicate will indeed reveal that the Prolog source has been consulted:

```
?- consult('/C:/Users/Admin/Desktop/hello.p').
Yes
?- listing.
hello :- write('Hello World!'), nl.
Yes
```

When the user predicate hello/0 is now called, it will in turn first call the system predicate write/1 and display the text “Hello World!”. Subsequently it will call the system predicate nl/0 which will issue a carriage return to the console:

```
?- hello.
Hello World!
Yes
```

We do not need to quite the runtime library to change the Prolog text. One can simply turn to the text editor again and continue editing the existing file hello.p. We want to change the hello/0 predicate into a hello/1 predicate:



```
hello(X) :- write('Hello '), write(X), nl.
```

Picture 4: Updating a Prolog Text

When the changed Prolog text has been saved, we can use the system predicate consult/1 to reread it. Before actually rereading the system predicate will remove all predicates it has previously read, so that hello/0 will disappear from the knowledge base.

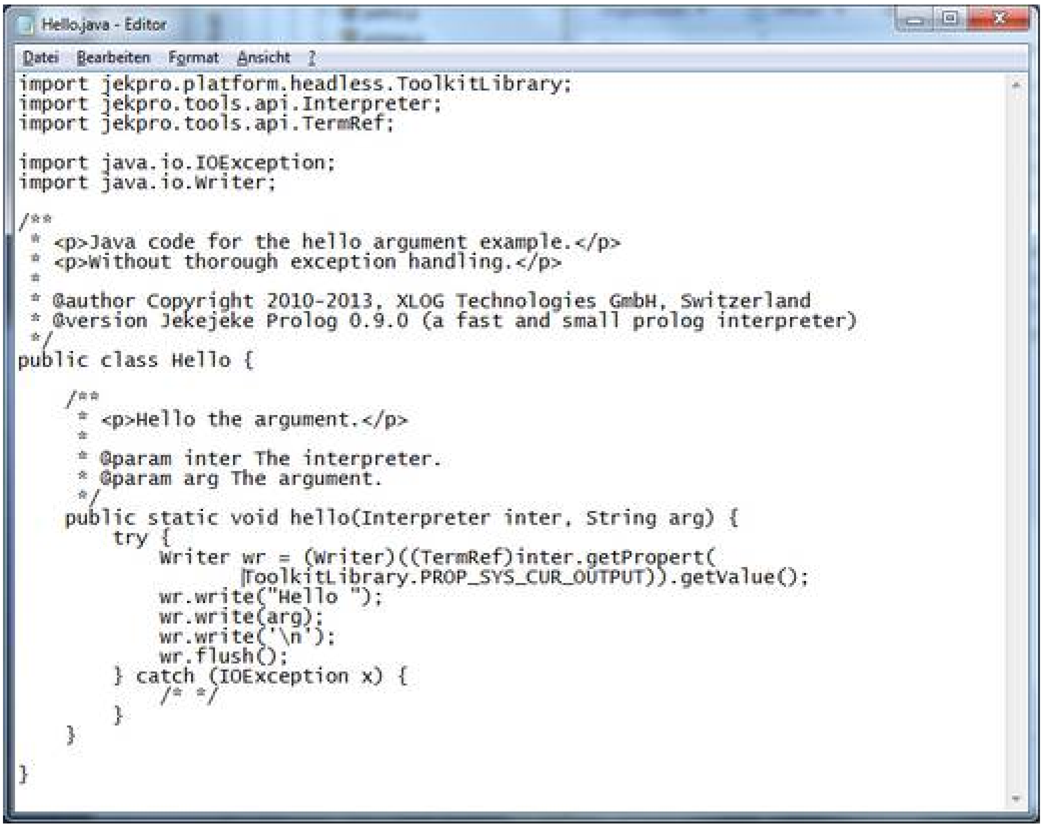
```
?- consult('/C:/Users/Admin/Desktop/hello.p').
Yes
?- listing.
hello(X) :- write('Hello '), write(X), nl.
Yes
```

The new predicate hello/1 will hello the given argument. We can test it directly in the console window of the runtime library:

```
?- hello('Anna').
Hello Anna
Yes
?- hello('Bert').
Hello Bert
Yes
```

2.2 Class Path Tour

In this guide we want to show how the runtime library can be connected with Java code. We will step by step show how a Java foreign predicate can be created and executed that will hello an argument. The Jekejeke Prolog runtime library does as well not provide an editor for Java classes. Again we have to refer to the given computing environment. We create the following Java class:



```java
import jekpro.platform.headless.ToolkitLibrary;
import jekpro.tools.api.Interpreter;
import jekpro.tools.api.TermRef;

import java.io.IOException;
import java.io.Writer;

/**
 * <p>Java code for the hello argument example.</p>
 * <p>Without thorough exception handling.</p>
 *
 * @author Copyright 2010-2013, XLOG Technologies GmbH, Switzerland
 * @version Jekejeke Prolog 0.9.0 (a fast and small prolog interpreter)
 */
public class Hello {

    /**
     * <p>Hello the argument.</p>
     *
     * @param inter The interpreter.
     * @param arg The argument.
     */
    public static void hello(Interpreter inter, String arg) {
        try {
            Writer wr = (Writer)((TermRef)inter.getPropert(
                    ToolkitLibrary.PROP_SYS_CUR_OUTPUT)).getValue();
            wr.write("Hello ");
            wr.write(arg);
            wr.write('\n');
            wr.flush();
        } catch (IOException x) {
            /* */
        }
    }

}
```

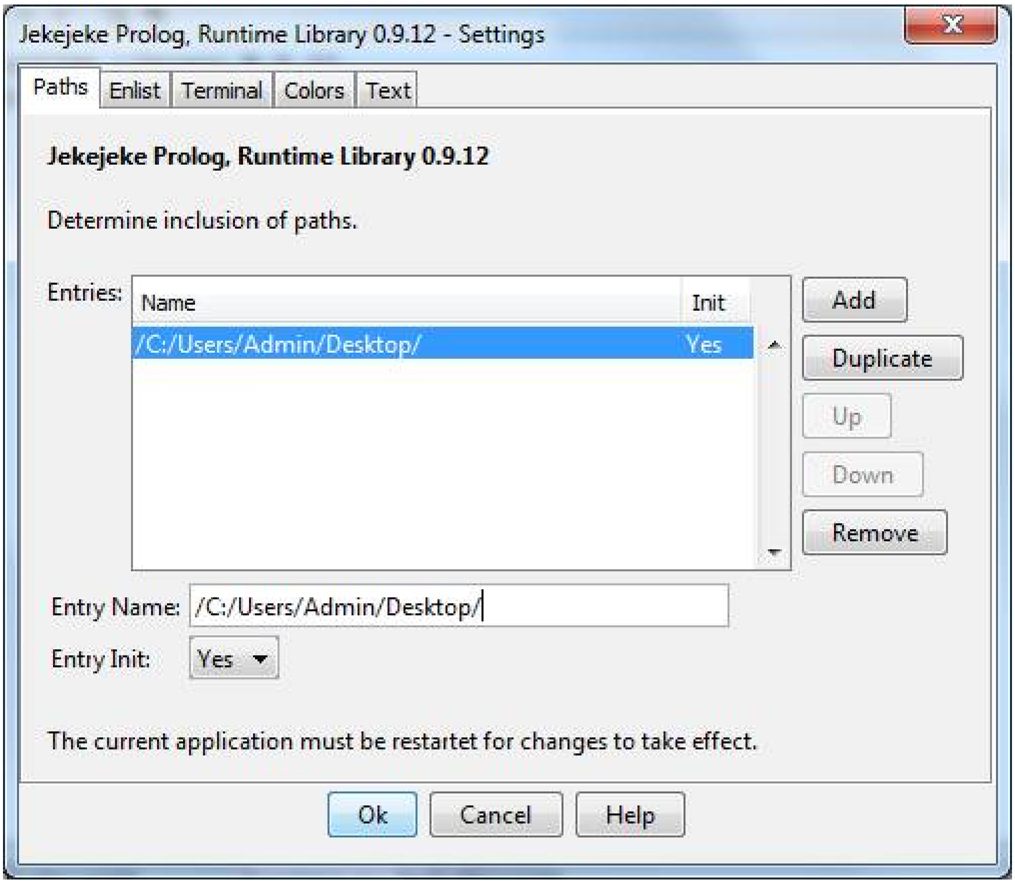
Picture 5: Creating a Java Class Source

The Java class provides a Java method hello. This method hellos the given argument on the output stream of the given interpreter. To be able to compile the java class the Jekejeke Prolog runtime library has to be present. For more details on compiling with the runtime library see the installation guide. When successful the Java compiler will have created a byte code file Hello.class, which will contain the Java method hello.

We can now start the Jekejeke Prolog runtime library and try to access the Java method hello. We will try to declare a new user predicate hello/1, which will be defined via the Java method hello. Our attempt will terminate with an exception, since the file Hello.class is not known to the Jekejeke Prolog runtime library per se:

```
?- foreign(hello/1, 'Hello', hello('Interpreter', 'String')).
Error: Class 'Hello' not found.
      foreign / 3
```

We can change the situation. We simply have to add the directory with the file Hello.class to the class path. This can be done via the class path dialog. Note the slash at the end of the path. It is needed, so that the class loader will know that the path points to a directory and not to an archive:



Jekejeke Prolog, Runtime Library 0.9.12 - Settings

Jekejeke Prolog, Runtime Library 0.9.12

Determine inclusion of paths.

Entries:

Name	Init
/C:/Users/Admin/Desktop/	Yes

Entry Name: /C:/Users/Admin/Desktop/

Entry Init: Yes

The current application must be restartet for changes to take effect.

Picture 6: Setting the Class Path

After having changed the class path, we need to restart the Jekejeke Prolog runtime library. The declaration of the Java foreign predicate should then succeed. We can use the system predicate listing/0 to check whether the Java foreign predicate was added to the current knowledge base:

```
        ?- foreign(hello/1, 'Hello', hello('Interpreter', 'String')).
        Yes
        ?- listing.
        :- foreign(hello / 1, 'Hello', hello('Interpreter', 'String')).
        Yes
```

The new predicate hello/1 will hello the given argument. Again we can test it directly in the console window of the runtime library:

```
        ?- hello('Anna').
        Hello Anna
        Yes
        ?- hello('Bert').
        Hello Bert
        Yes
```

Since the ISO core standard requires that the current output may also point to a binary stream, we cannot guarantee that it is of type Writer. The Java method for the hello predicate is thus not safe from class cast exceptions. Also the current code suppresses an eventual IOException thrown by the character output. The class Hello2 in the appendix shows an alternative implementation. The alternative code does check the type of the current output and it passes on an IOException in the form of an InterpreterMessage.

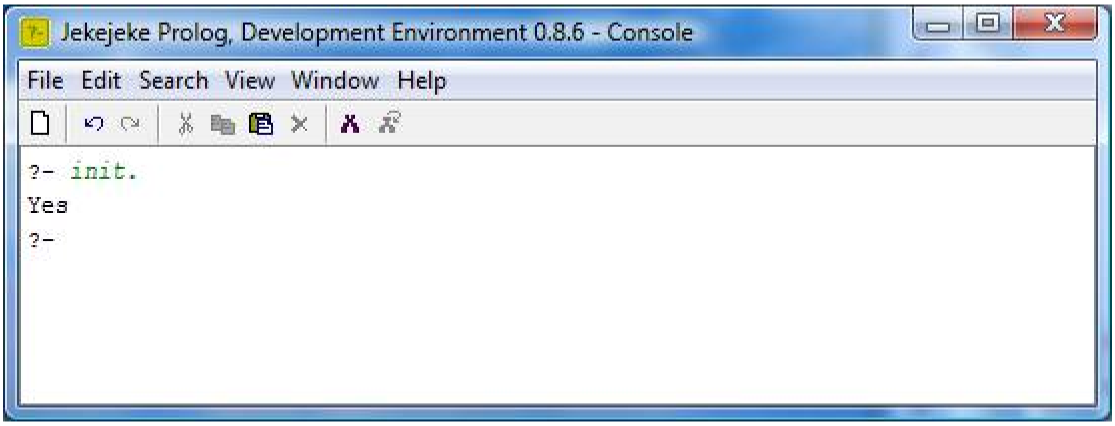
2.3 Threads Tour

In this guide we want to show how it is possible to execute easily multiple threads in the runtime library. The end-user can open multiple console windows. Each console window has a Java thread associated that executes a Jekejeke Prolog interpreter which in turn executes a Prolog query answer loop. Each Jekejeke Prolog interpreter has a standard input and standard output stream which are associated with the console. These streams are thread safe and can thus be accessed by multiple threads. We will use one standard output stream as a logging device for our multiple threads demonstration.

The logging device can be initialized by the following init predicate:

```
      init :- current_output(X), assertz(console(X)).
```

The init predicate will first access the standard output of the Jekejeke Prolog interpreter that executes the predicate. And then asserts the corresponding stream as a console fact. Accessing facts and rules is also thread safe so that the console fact can be later accessed by multiple threads. To init the logging device we use the main window that comes up when starting up the interpreter. That we are in the main window is indicated by the name “Console” in the window title. The consulting of the Prolog text is not shown:


Jekejeke Prolog, Development Environment 0.8.6 - Console
File Edit Search View Window Help

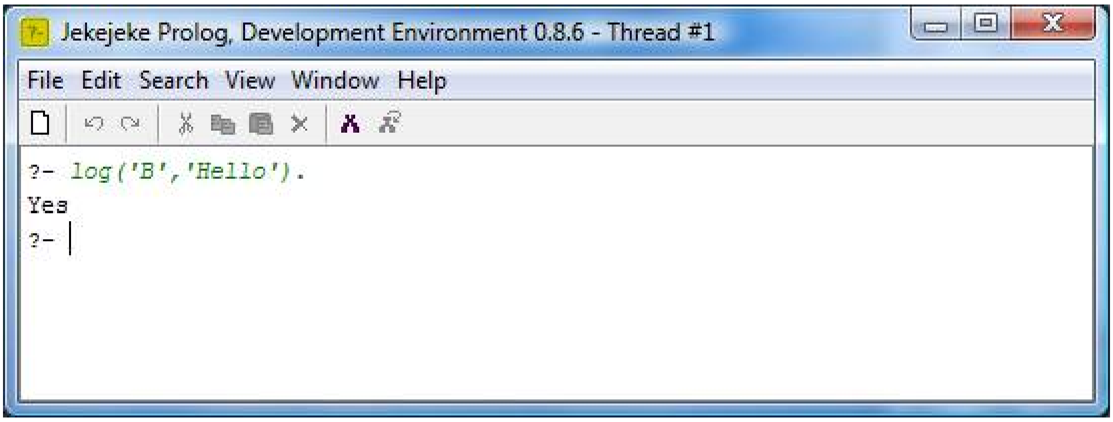
```
?- init.
Yes
?-
```

Picture 7: Initializing the Logging Device

A new console window and thus thread can be created by the menu item File | New… An additional console window can be recognized by the name “Thread #<Number>” in the window title. The “<Number>” is the running number of the additional console windows. To test whether we can write into the main window, we use the following additional log predicate:

```
      log(X,Y) :-
          atom_concat('Process ',X,A1),
          atom_concat(A1,': ',A2),
          atom_concat(A2,Y,A3),
          atom_concat(A3,'\n',A4),
          console(Z),
          write(Z,A4),
          flush_output(Z).
```

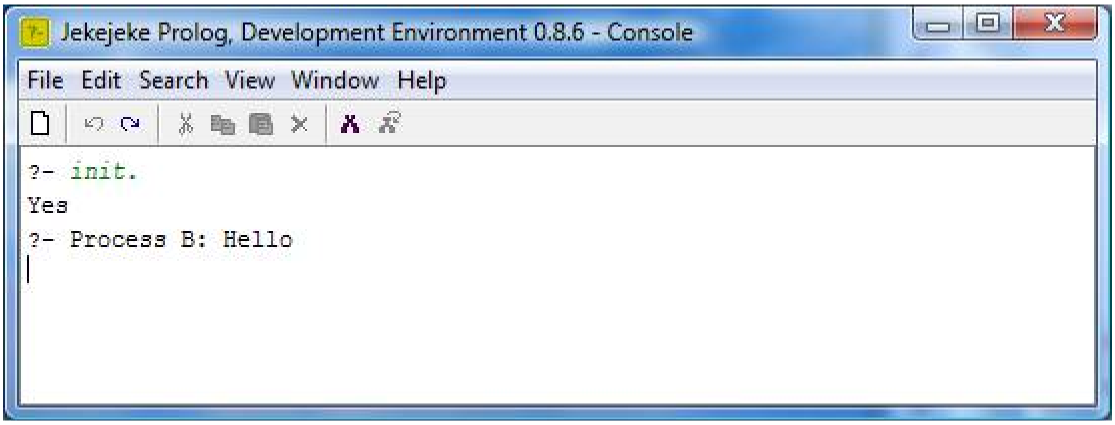
This predicate will first prepare an atom with the desired text and then write it in one go to the logging device. Since the logging device writes to the main window we will not see it in the additional console window. All that can be seen in the additional console window is that the log predicate succeeds:


Jekejeke Prolog, Development Environment 0.8.6 - Thread #1
File Edit Search View Window Help

```
?- log('B','Hello').
Yes
?-
```

Picture 8: Writing to the Logging Device

But when we turn to the main window, we see the text that has been logged to the logging device which is the standard output of the main window. We see the text as follows:


Jekejeke Prolog, Development Environment 0.8.6 - Console
File Edit Search View Window Help

```
?- init.
Yes
?- Process B: Hello
```

Picture 9: Display of a Logging Message

We will now go on and define a perpetual process. There are different approaches to realize perpetual process in Prolog. One approach consists of using argument variables to carry around a state. The programming pattern looks as follows:

```
process(S1) :-
      transition(S1,S2),
      process(S2).
```

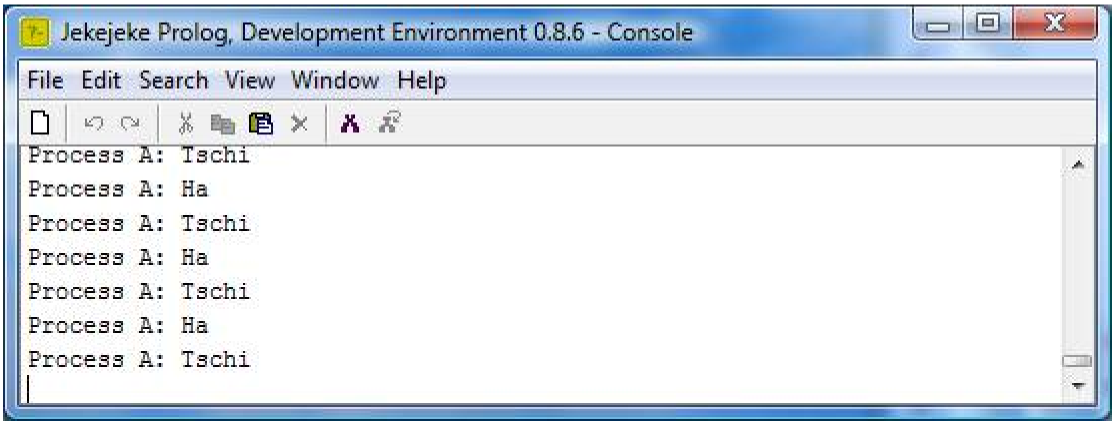
This approach does not yet fully work in Jekejeke Prolog. The stack frame elimination will keep the number of choice points constant when the predicate transition/1 is deterministic. But the variable counter will currently increment indefinitely, since we do not yet have implemented a variable counter compression method. This leads to inconsistencies in the lexical comparison of variables and in the output of variables. Therefore we propose a method that uses the database for the state. The programming pattern looks as follows:

```
process(ID) :-
      repeat,
      retract(state(ID,S1)),
      transition(S1,S2),
      assertz(state(ID,S2)),
      fail.
```

The second programming pattern works since the predicate repeat/0 does not increment the variable counter. The example we are using here is even simpler. We don’t have a state at all but use still the repeat programming pattern to avoid the variable counter problem. Our example perpetual process will indefinitely write “Ha” and “Tschi” on the logging device. It is defined as follows:

```
process(X) :-
    repeat,
    log(X,'Ha'),
    log(X,'Tschi'),
    fail.
```

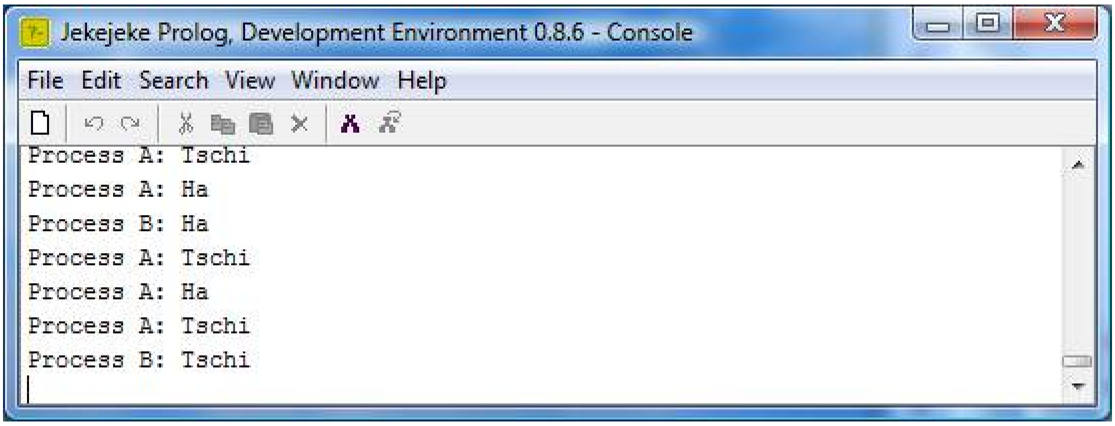
We can start the example perpetual process via process(‘A’) on the main window. We will see that the main console is indefinitely filled with the logging of “Ha” and “Tschi”. The console content will not grow indefinitely since it is itself a first-in-last-out buffer. Also the console will not fill very quickly since the output is throttled. Both parameters, the buffer length and the scroll delay, can be set in the terminal settings.


Jekejeke Prolog, Development Environment 0.8.6 - Console
File Edit Search View Window Help

```
Process A: Tschi
Process A: Ha
Process A: Tschi
Process A: Ha
Process A: Tschi
Process A: Ha
Process A: Tschi
```

Picture 10: Process process(‚A’) was started

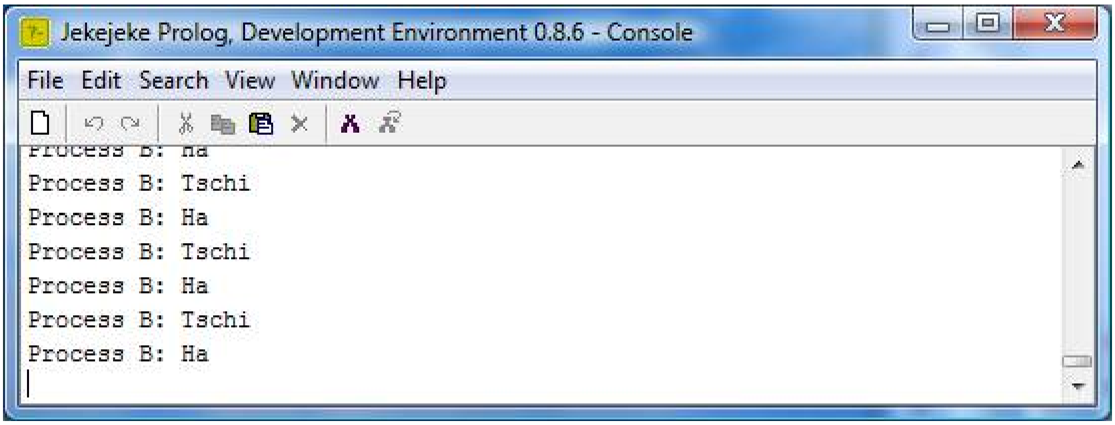
We can now start an additional example perpetual process via process(‘B’) in the additional window. This process will also fill the main window since the main window is our logging device. There is no fixed order when the two processes will write in the logging device. They are true Java threads and we will observe true interleaving. For example in the below screenshot we see that between the ‘Ha’ of process B and the ‘Tschi’ of process B, the process A is able to issue 3 write instructions.


Jekejeke Prolog, Development Environment 0.8.6 - Console
File Edit Search View Window Help

```
Process A: Tschi
Process A: Ha
Process B: Ha
Process A: Tschi
Process A: Ha
Process A: Tschi
Process B: Tschi
```

Picture 11: Process process(‚B’) was also started

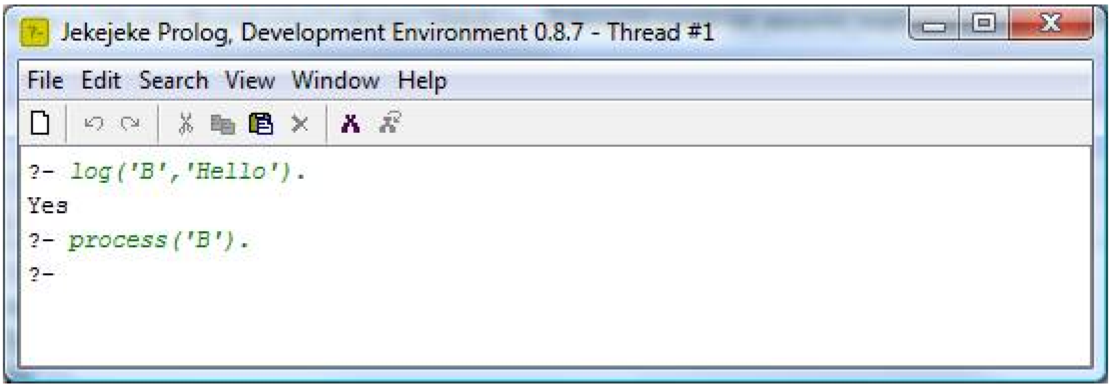
The standard output can be put on hold anytime by pressing Ctrl-S and resumed by pressing Ctrl-S again. When the standard output is put on hold, the current write instruction on this stream will be blocked and subsequently all other threads also trying to execute a write instruction on this stream will be also blocked. Thread execution can be stopped by invoking the menu item File | Abort. When we do this on the main window, the process('A') will be aborted, but the process('B') will still continue:


Jekejeke Prolog, Development Environment 0.8.6 - Console
File Edit Search View Window Help

```
Process B: Ha
Process B: Tschi
Process B: Ha
Process B: Tschi
Process B: Ha
Process B: Tschi
Process B: Ha
```

Picture 12: Process process(‚A') was aborted

To stop the thread of the process('B') we have to issue the menu item File | Abort from the additional window. In our example when we do this, we will not have any thread anymore writing to the logging device and thus the main window will turn silent. The menu item File | Abort thus nothing else than inject an exception to the running process. The particular exception is handled by the query answer loop. Therefore we will not see the exception message or its context:


Jekejeke Prolog, Development Environment 0.8.7 - Thread #1
File Edit Search View Window Help

```
?- log('B','Hello').
Yes
?- process('B').
?-
```

Picture 13: Process process(‚B') is also aborted

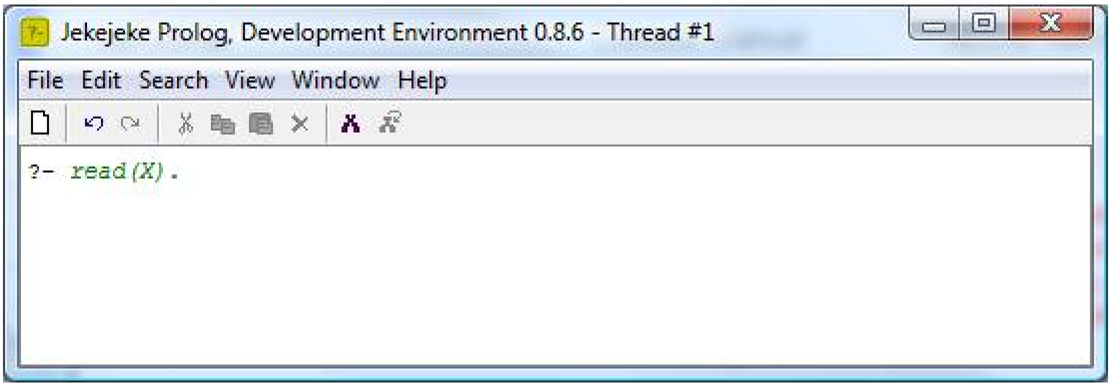
Aborting a thread is irreversible. Alternatively the menu item File | Trace can be used to put the thread into trace mode. This will allow to inspect the currently executing instructions in more detail and to decide whether an abort is really necessary.

2.4 Memory Low Tour

The runtime library uses the Java memory manager to check whether the used memory exceeds a threshold. Currently a threshold of 85% of the maximum memory granted to the runtime library has been hard-coded. When the threshold is exceeded the runtime library will signal a memory low to all console threads. Threads not associated with a Jekejeke console window are not signalled.

We can distinguish two important modes of a thread. Either a thread is running and executing code. Or a thread is blocking and waiting for an event. To indicate the existence of a signal the runtime library will also interrupt the console threads. This allows for the console threads to escape the blocking. We will demonstrate in the following how both a blocking and non-blocking console thread are signalled memory low.

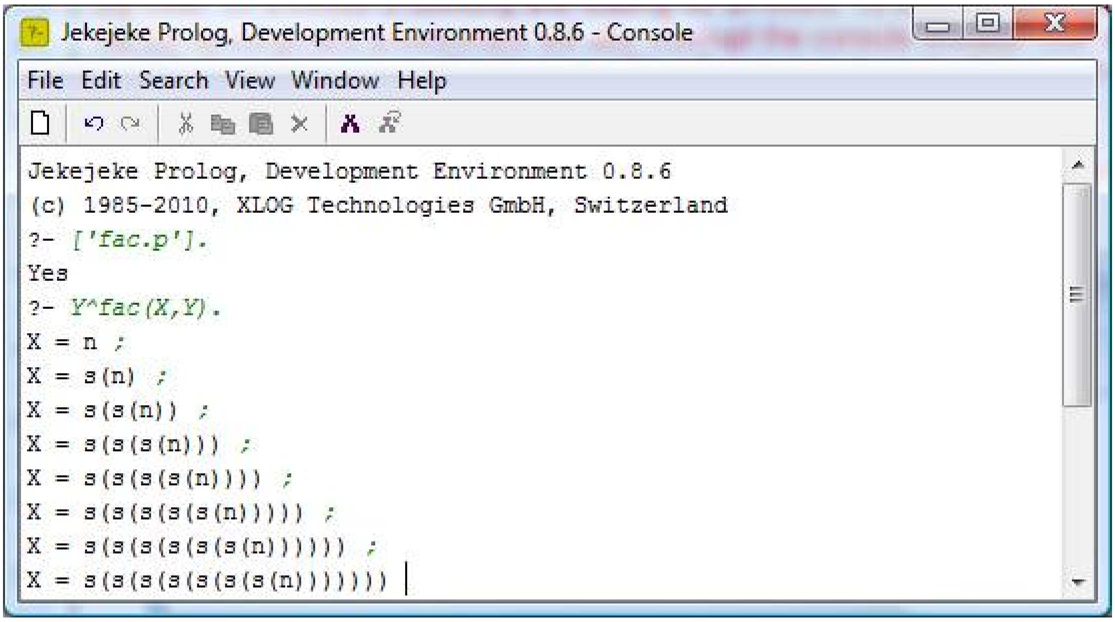
For the blocking thread we open an additional console window. We simply execute a read statement, which will normally block until the end-user has given some text:


Jekejeke Prolog, Development Environment 0.8.6 - Thread #1
File Edit Search View Window Help

```
?- read(X).
```

Picture 14: The Blocking Thread

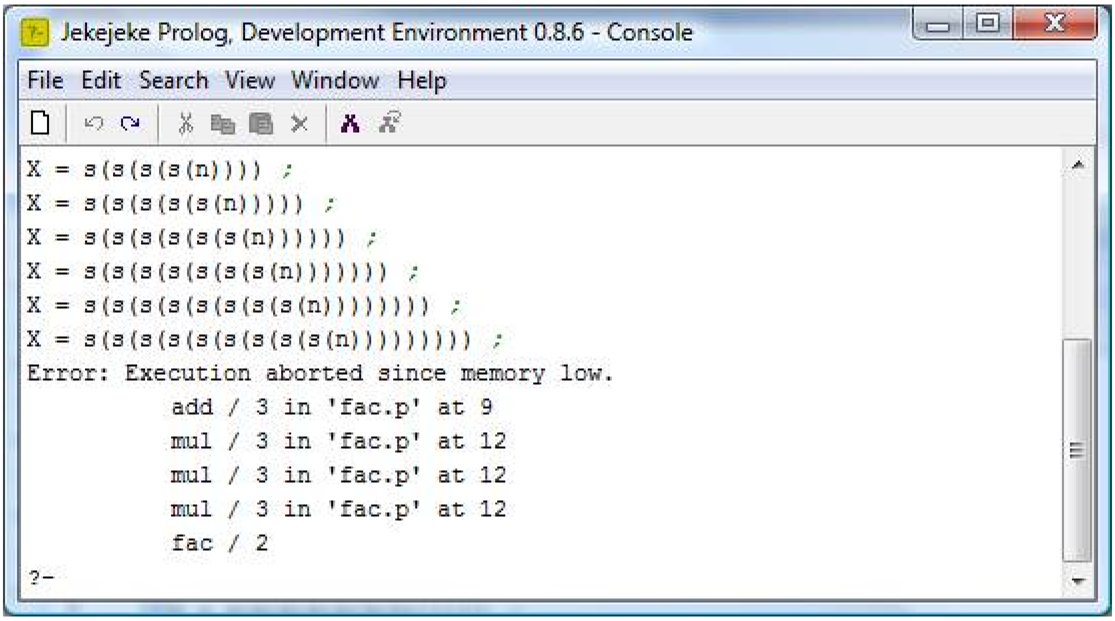
For the non-blocking thread we will compute the factorial via the Peano axioms. The Prolog text for the factorial is given in the appendix.


Jekejeke Prolog, Development Environment 0.8.6 - Console
File Edit Search View Window Help

```
Jekejeke Prolog, Development Environment 0.8.6
(c) 1985-2010, XLOG Technologies GmbH, Switzerland
?- ['fac.p'].
Yes
?- Y^fac(X,Y).
X = n ;
X = s(n) ;
X = s(s(n)) ;
X = s(s(s(n))) ;
X = s(s(s(s(n)))) ;
X = s(s(s(s(s(n))))) ;
X = s(s(s(s(s(s(n)))))) ;
X = s(s(s(s(s(s(s(n)))))))
```

Picture 15: The Non-Blocking Thread

Each subsequent answer will gradually need more memory to produce its result. Eventually the threshold will be reached and a memory low error will be raised:

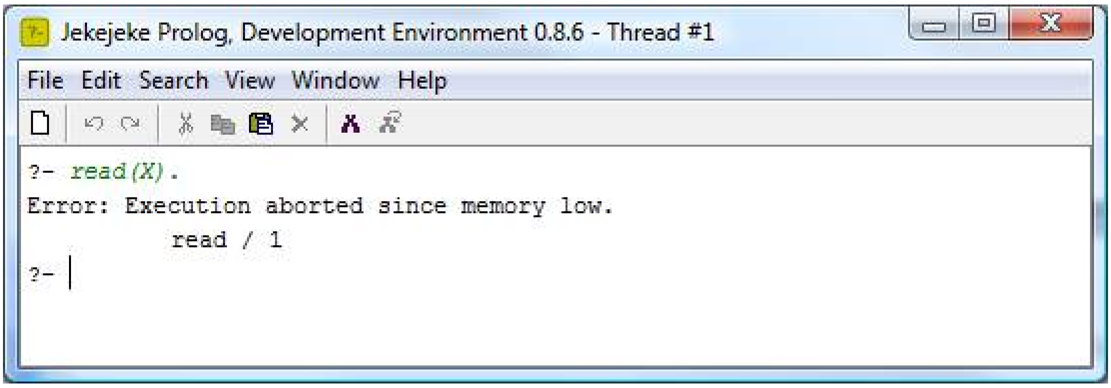


Jekejeke Prolog, Development Environment 0.8.6 - Console
File Edit Search View Window Help

```
X = s(s(s(s(n)))) ;
X = s(s(s(s(s(n))))) ;
X = s(s(s(s(s(s(n)))))) ;
X = s(s(s(s(s(s(s(n))))))) ;
X = s(s(s(s(s(s(s(s(n)))))))) ;
X = s(s(s(s(s(s(s(s(s(n))))))))) ;
Error: Execution aborted since memory low.
          add / 3 in 'fac.p' at 9
          mul / 3 in 'fac.p' at 12
          mul / 3 in 'fac.p' at 12
          mul / 3 in 'fac.p' at 12
          fac / 2
?-
```

Picture 16: Error shown by the First Thread

The runtime library will not bother with determining a victim. The current implementation is such that it will signal all console threads blindly. As a result the second thread will also abort with a memory low error:



Jekejeke Prolog, Development Environment 0.8.6 - Thread #1
File Edit Search View Window Help

```
?- read(X).
Error: Execution aborted since memory low.
          read / 1
?-
```

Picture 17: Error shown by the Second Thread

Although all console threads are signalled at the same moment, it might take some time until a console thread shows the error message. When a Jekejeke Prolog thread has accepted a signal it will throw the signal as an error. The thread will then first be busy with undoing its data structures and the execution of eventual catch blocks.

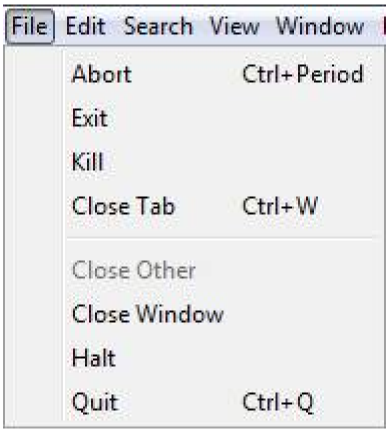
3 Menus

The Jekejeke Prolog runtime library does not provide the editing of programs. For this purpose arbitrary external editors can be used. Even editors that come with external integrated development environment are suitable. Nevertheless some action items to control the interpreter are needed. They primarily come as menu items.

The following menus are available:

- File Menu
- Edit Menu
- Search Menu
- View Menu
- Build Menu
- Window Menu
- Help Menu

3.1 File Menu



Abort: Abort the current thread. The associated interpreter will return to the current query answer loop.

Exit: Exit the current thread. The associated interpreter will return to the previous query answer loop.

Kill: Kill the current thread. The associated interpreter will immediately stop execution.

Close Tab: Close the current thread. The associated interpreter will exit all query loops.

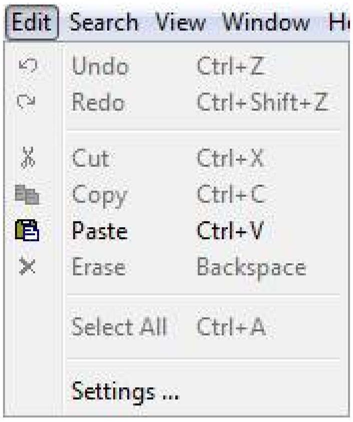
Close Other: Close all threads of this window different from the current thread.

Close Window: Close all threads of this window.

Halt: Halt this application.

Quit: Close all windows of this application.

3.2 Edit Menu



Undo: Undoes the last edit operation.

Redo: Redoes the last edit operation.

Cut: Copies the selected region into the edit buffer and deletes the selected region.

Copy: Copies the selected region into the edit buffer.

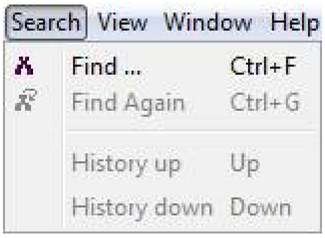
Paste: Deletes the selected region and inserts the edit buffer.

Erase: Deletes the selected region or deletes the character to the left.

Select All: Selects the whole text area.

Settings ...: Invokes the settings dialogs. See also Settings Panels.

3.3 Search Menu



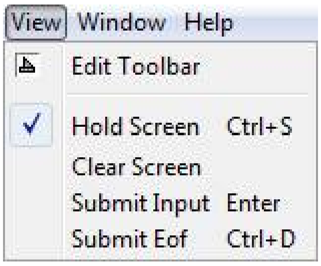
Find …: Invokes the search dialog. See also Search Dialog.

Find Again: Searches the last search string again.

History up: Displays the next command up in the history.

History down: Displays the next command down in the history.

3.4 View Menu



Edit Toolbar: Shows or hides the edit toolbar. See also Edit Toolbar.

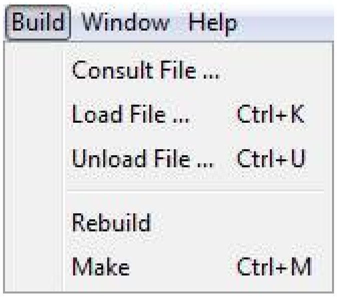
Hold Screen: Suspend resp. resume the output of the interpreter.

Clear Screen: Clear the output of the interpreter.

Submit Input: Submit the current input to the interpreter.

Submit Eof: Submit an end-of-file to the interpreter.

3.5 Build Menu



Consult File …: Select a file and consult it.

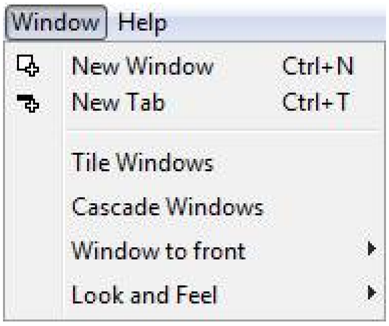
Load File …: Select a file and ensure load it.

Unload File …: Select a file and detach it.

Rebuild: Consult all used files.

Make: Ensure load all used files.

3.6 Window Menu



New Window: Start a new thread in a new window.

New Tab: Start a new thread in a new tab.

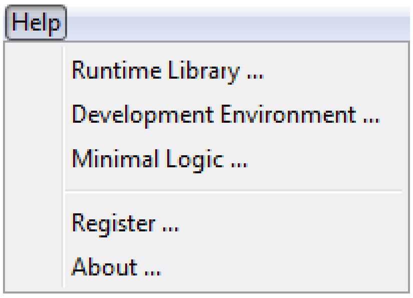
Tile Windows: Tile the console windows.

Cascade Windows: Cascade the console windows.

Window to Front: Bring a console window to front.

Look and Feel: Change the look and feel of the console windows.

3.7 Help Menu



<Capability> …: Opens the documentation of the predefined or enlisted capability.

Register ...: Invokes the register dialog. See also Register Dialog.

About …: Invokes the about dialog. See also About Dialog.

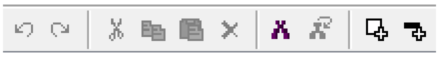
4 Toolbars and Popup Menus

Toolbars can be attached to windows and allow the invocation of action items quicker than menus. Therefore we also provide access to some action items via the toolbar. Further improvements in usability are pop-up menus that can be invoked by right-click mouse.

The following toolbars and popup menus are available:

- Edit Toolbar
- Edit Popup Menu

4.1 Edit Toolbar



Undo: Undoes the last edit operation.

Redo: Redoes the last edit operation.

Cut: Copies the selected region into the edit buffer and deletes the selected region.

Copy: Copies the selected region into the edit buffer.

Paste: Deletes the selected region and inserts the edit buffer.

Erase: Deletes the selected region.

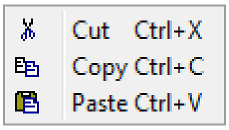
Find …: Invokes the search dialog. See also Search Dialog.

Find Again: Searches the last search string again.

New Window: Start a new thread in a new window.

New Tab: Start a new thread in a new tab.

4.2 Edit Popup Menu



Cut: Copies the selected region into the edit buffer and deletes the selected region.

Copy: Copies the selected region into the edit buffer.

Paste: Deletes the selected region and inserts the edit buffer.

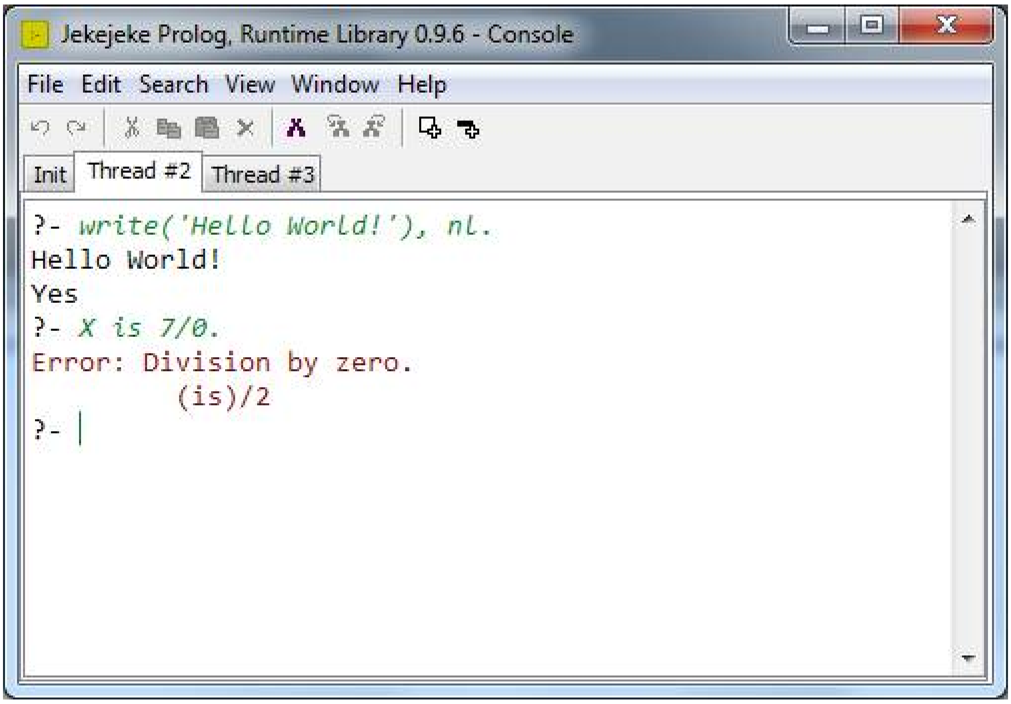
5 Console Windows & Dialogs

The console consists of a terminal window for the interaction with a Jekejeke Prolog thread. Besides a text dialog based interaction it is also possible to interact via the menus, the toolbar and the pop up menu with the Jekejeke Prolog thread.

The following console windows & dialogs are available:

- Terminal Window
- Search Dialog

5.1 Terminal Window


Jekejeke Prolog, Runtime Library 0.9.6 - Console
File Edit Search View Window Help
Init
Thread #2
Thread #3

```
?- write('Hello World!'), nl.
Hello World!
Yes
?- X is 7/0.
Error: Division by zero.
         (is)/2
?- 
```

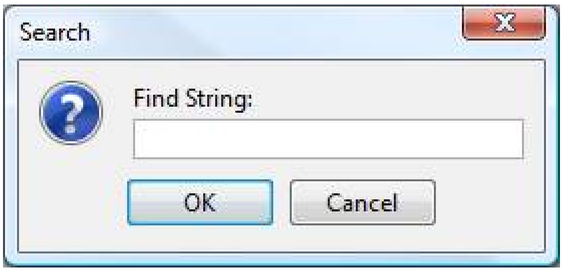
Menu bar: The menus. See also Menus.

Toolbar: The toolbar. See also Edit Toolbar.

Tab Widget: Tabs to select a thread.

Text Area: The text input and output area. See also Edit Popup Menu.

5.2 Search Dialog



Find String: The string to search.

Ok: Search the string and close the dialog.

Cancel: Close the dialog.

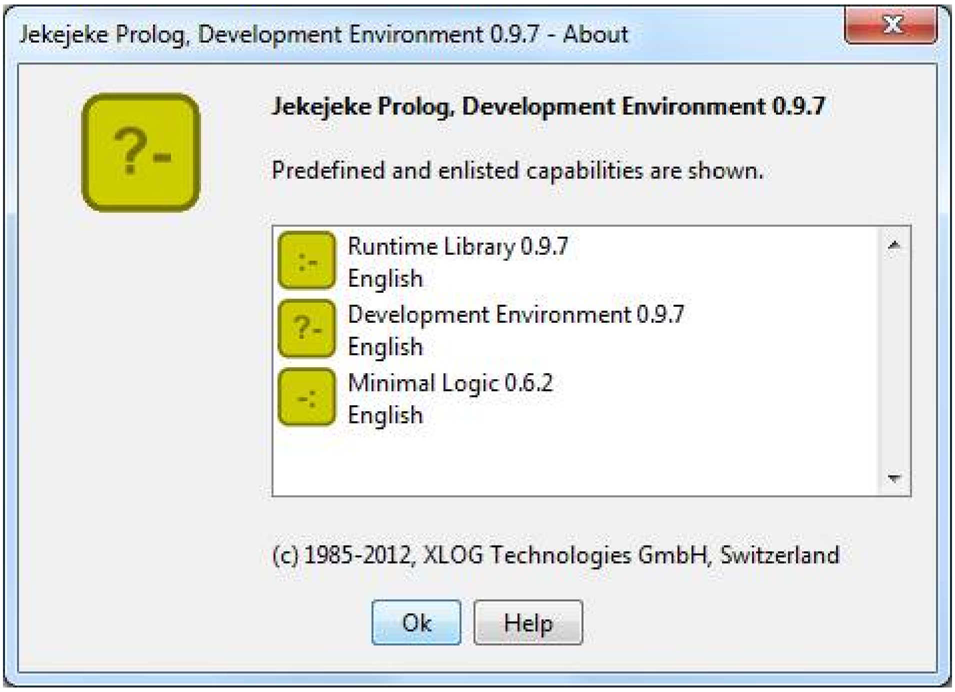
6 Activation Dialogs & Panels

The activation dialogs & panels deal with the management of the capability store. Only enlisted capabilities are shown. It is possible to interactively activate capabilities.

The following activation dialogs & panels are available:

- About Dialog
- Register Dialog
- Information Panel
- Service Panel
- Email Dialog

6.1 About Dialog



Jekejeke Prolog, Development Environment 0.9.7 - About

Jekejeke Prolog, Development Environment 0.9.7

Predefined and enlisted capabilities are shown.

Runtime Library 0.9.7
English
Development Environment 0.9.7
English
Minimal Logic 0.6.2
English

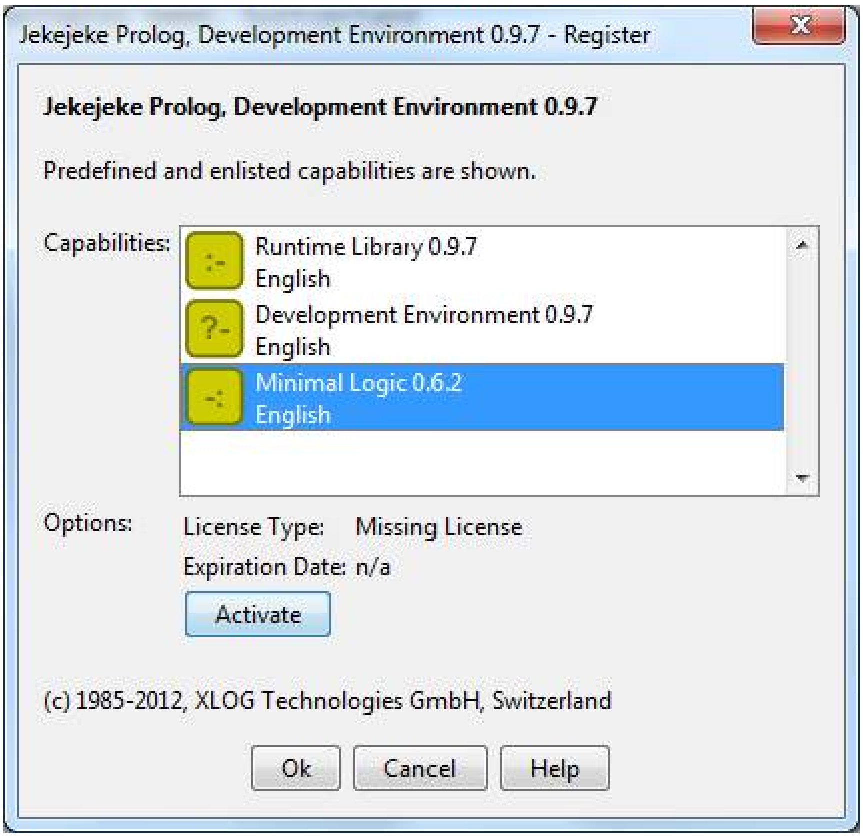
(c) 1985-2012, XLOG Technologies GmbH, Switzerland

List: The list of capabilities. See also Information Panel.

Ok: Close the dialog.

Help: Shows this help section.

6.2 Register Dialog



Jekejeke Prolog, Development Environment 0.9.7 - Register

Jekejeke Prolog, Development Environment 0.9.7

Predefined and enlisted capabilities are shown.

Capabilities:

- Runtime Library 0.9.7 English
- Development Environment 0.9.7 English
- Minimal Logic 0.6.2 English

Options: License Type: Missing License

Expiration Date: n/a

(c) 1985-2012, XLOG Technologies GmbH, Switzerland

Capabilities: The list of capabilities and the selected capability. See also Information Panel.

License Type: The license type of the selected capability.

Expiration Date: The expiration date of the selected capability.

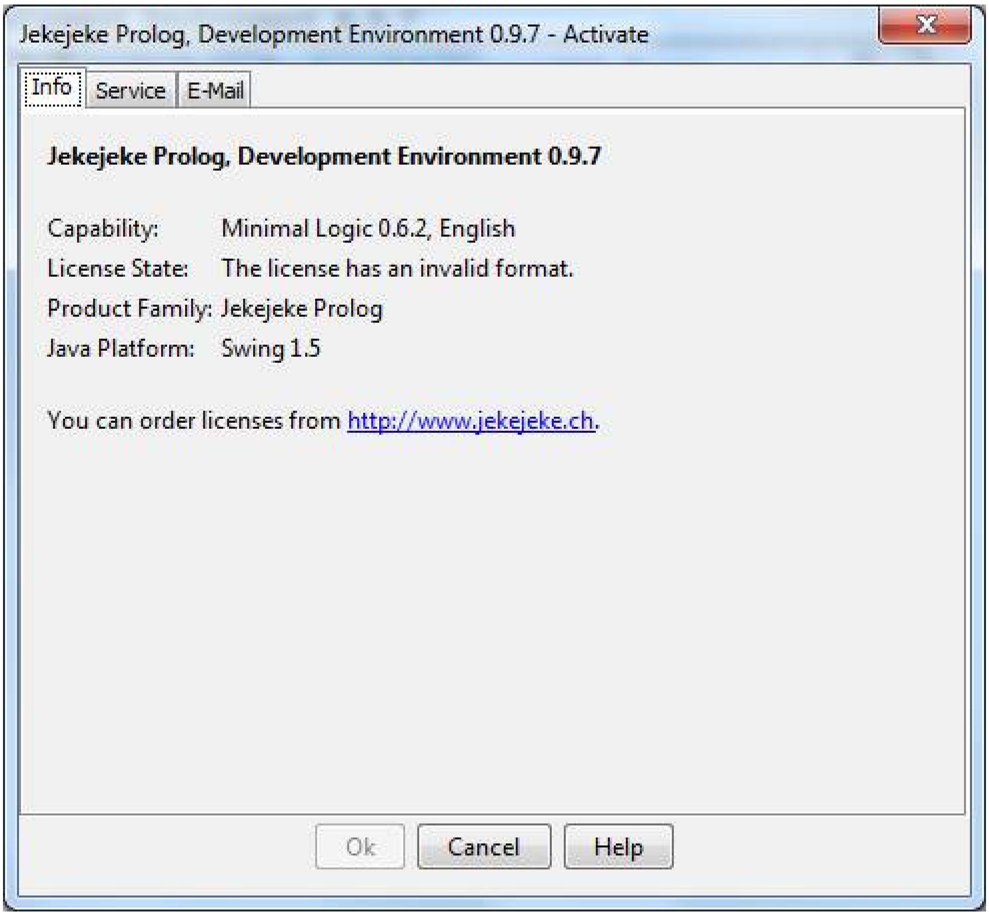
Activate: Activate the selected capability. See also Service Panel and Email Panel.

Ok: Verify completion and close the dialog.

Cancel: Close the dialog.

Help: Shows this help section.

6.3 Information Panel



Jekejeke Prolog, Development Environment 0.9.7

Capability: Minimal Logic 0.6.2, English
License State: The license has an invalid format.
Product Family: Jekejeke Prolog
Java Platform: Swing 1.5

You can order licenses from http://www.jekejeke.ch.

Capability: The selected capability.

License Status: The status of the license of the selected capability.

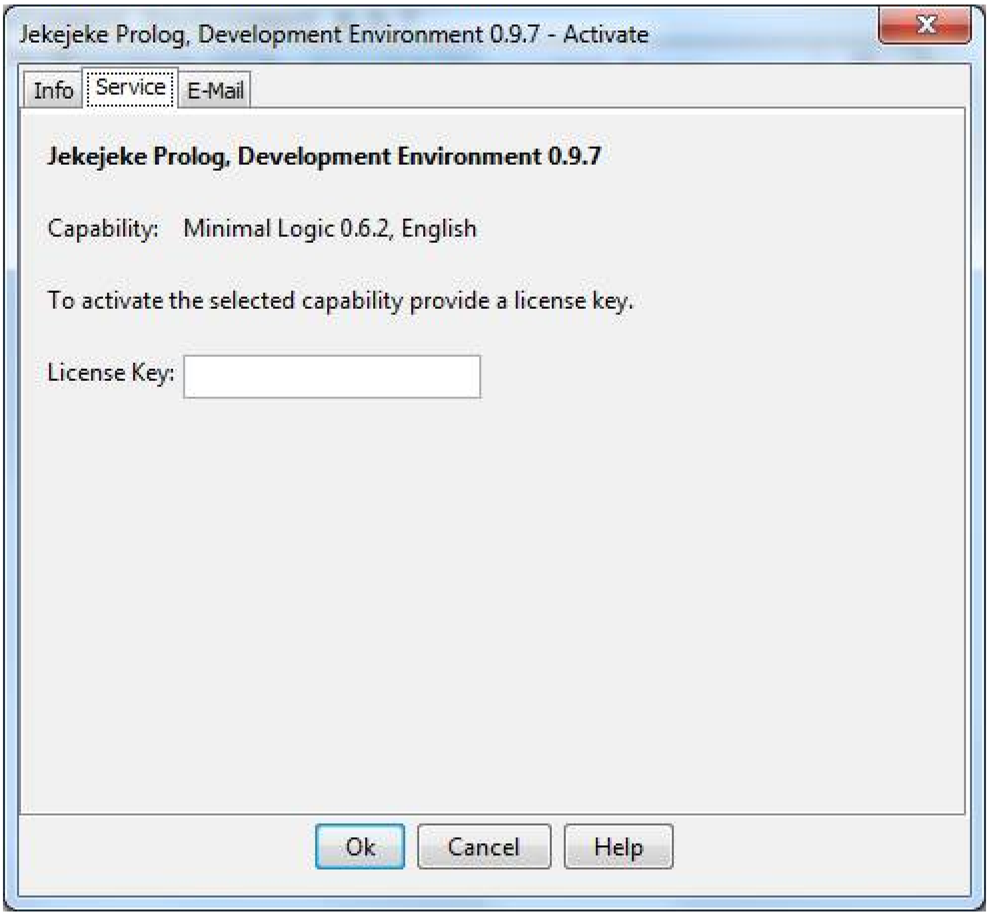
Product Family: The product family of the selected capability.

Java Platform: The required platform of the selected capability.

Cancel: Close the dialog.

Help: Shows this help section.

6.4 Service Panel



Jekejeke Prolog, Development Environment 0.9.7 - Activate

Jekejeke Prolog, Development Environment 0.9.7

Capability: Minimal Logic 0.6.2, English

To activate the selected capability provide a license key.

License Key:

Capability: The selected capability.

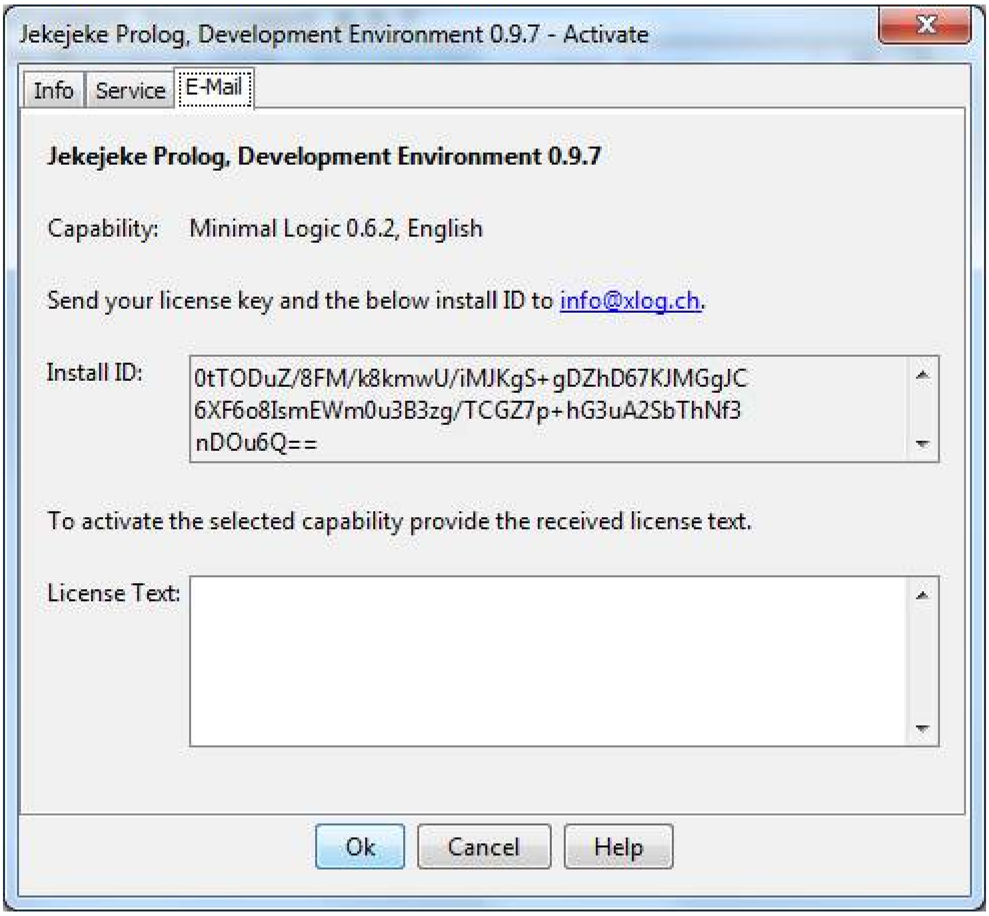
License Key: The key for the activation for the license.

Ok: Activate the license, verify the license and close the dialog.

Cancel: Close the dialog.

Help: Shows this help section.

6.5 Email Panel



Jekejeke Prolog, Development Environment 0.9.7

Capability: Minimal Logic 0.6.2, English

Send your license key and the below install ID to info@xlog.ch.

Install ID: 0tTODuZ/8FM/k8kmwU/iMJKgS+gDZhD67KJMGgJC6XF6o8IsmEWm0u3B3zg/TCGZ7p+hG3uA2SbThNf3nDOu6Q==

To activate the selected capability provide the received license text.

License Text:

Capability: The selected capability.

Install ID: The installation ID to send.

License Text: The received text for the license.

Ok: Register the license text, verify the license and close the dialog.

Cancel: Close the dialog.

Help: Shows this help section.

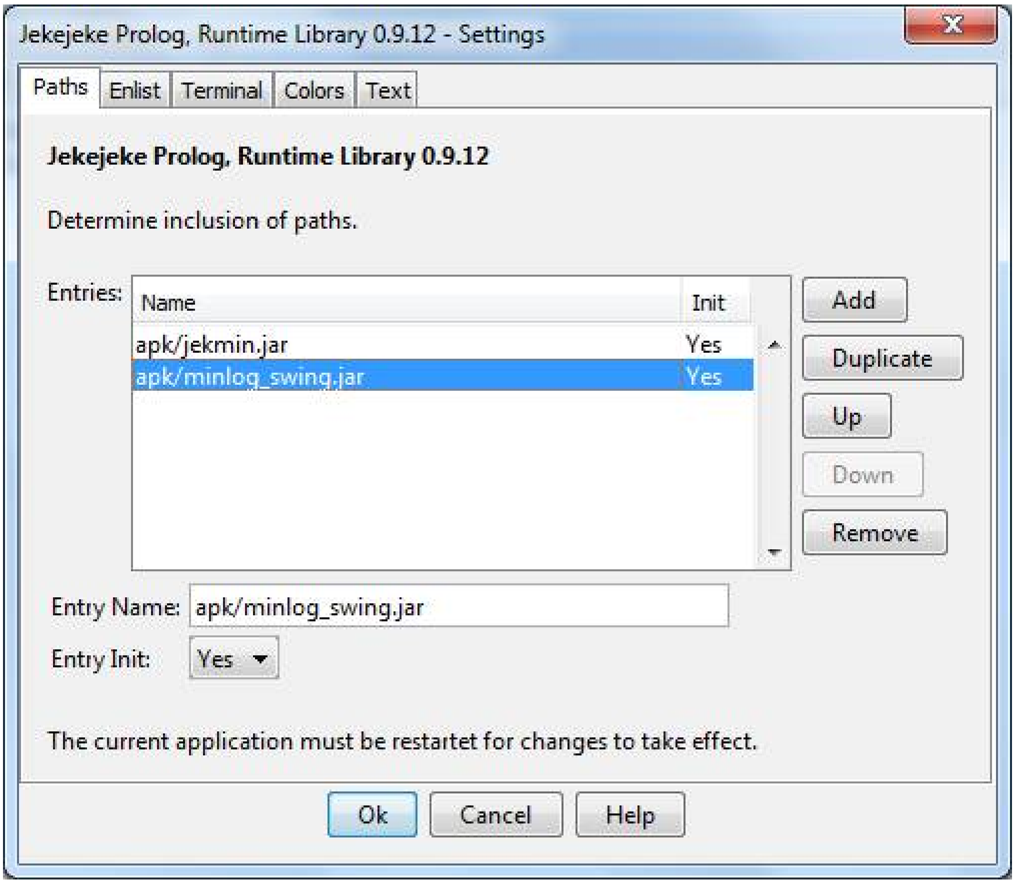
7 Settings Panels

The settings dialog shows tabbed panels. Each panel is responsible for a particular set of settings. All the settings are stored in the user profile. Some changes are only effective after a restart of the runtime library.

The following settings panels are available:

- Class Path Panel
- Enlist Panel
- Terminal Panel
- Colours Panel
- Text Panel
- Language Panel

7.1 Class Path Panel



Jekejeke Prolog, Runtime Library 0.9.12 - Settings

Jekejeke Prolog, Runtime Library 0.9.12

Determine inclusion of paths.

Entries:

Name	Init
apk/jekmin.jar	Yes
apk/minlog_swing.jar	Yes

Entry Name: apk/minlog_swing.jar

Entry Init: Yes

The current application must be restartet for changes to take effect.

Entries: The paths to be included.

Add: Add a new path.

Duplicate: Duplicate the selected path.

Up: Move the selected path up.

Down: Move the selected path down.

Remove: Remove the selected path.

Entry Name: The archive or directory of the selected path.

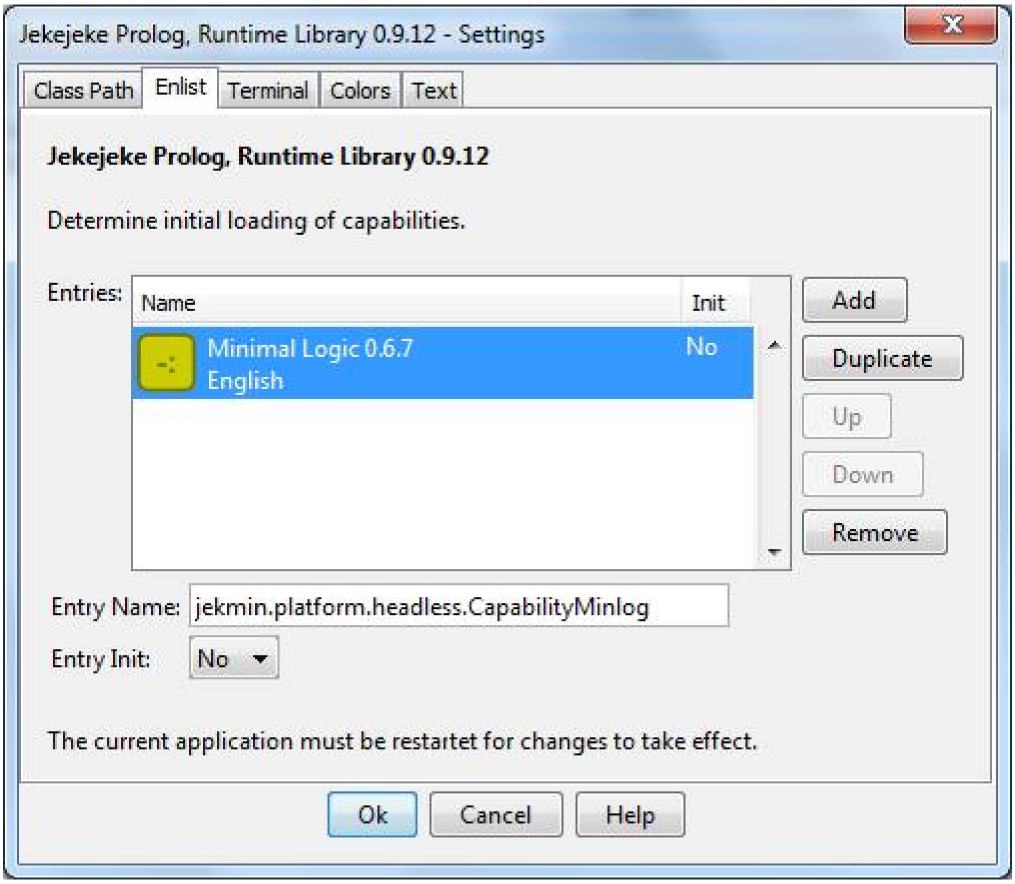
Entry Init: The inclusion mode of the capability.

Ok: Save the settings and close the dialog.

Cancel: Close the dialog.

Help: Show this help section.

7.2 Enlist Panel



Jekejeke Prolog, Runtime Library 0.9.12 - Settings

Jekejeke Prolog, Runtime Library 0.9.12

Determine initial loading of capabilities.

Entries:

Name	Init
Minimal Logic 0.6.7 English	No

Entry Name: jekmin.platform.headless.CapabilityMinlog

Entry Init: No

The current application must be restartet for changes to take effect.

Entries: The capabilities to be loaded.

Add: Add a new capability.

Duplicate: Duplicate the selected capability.

Up: Move the selected capability up.

Down: Move the selected capability down.

Remove: Remove the selected capability.

Entry Name: The class name of the selected capability.

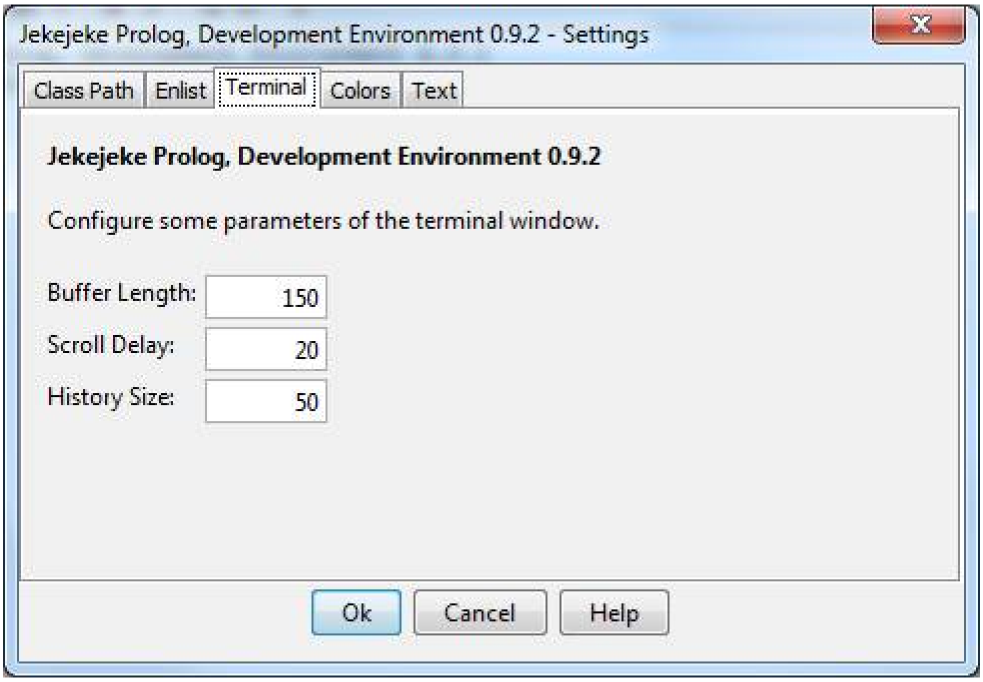
Entry Init: The initialization mode of the capability.

Ok: Save the settings and close the dialog.

Cancel: Close the dialog.

Help: Shows this help section.

7.3 Terminal Panel


Jekejeke Prolog, Development Environment 0.9.2 - Settings

Jekejeke Prolog, Development Environment 0.9.2

Configure some parameters of the terminal window.

Buffer Length: 150

Scroll Delay: 20

History Size: 50

Buffer Length: The maximum number of shown output lines from the interpreter.

Scroll Delay: The delay in milliseconds per output line of the interpreter.

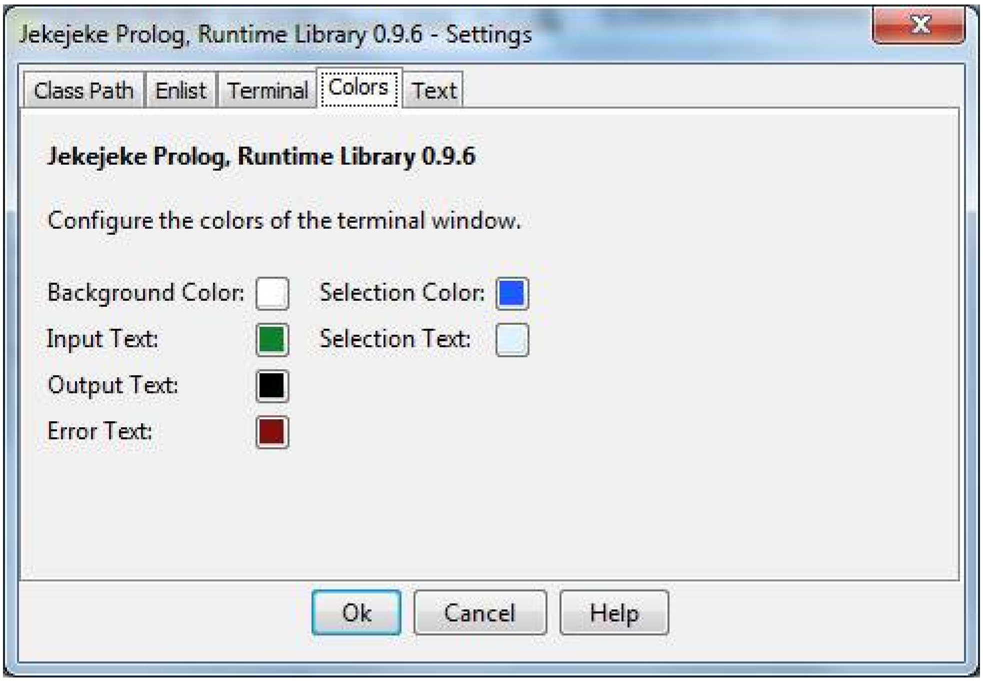
History Size: The maximum number of retained input lines from the end-user.

Ok: Apply the settings, save the settings and close the dialog.

Cancel: Close the dialog.

Help: Show this help section.

7.4 Colours Panel



Jekejeke Prolog, Runtime Library 0.9.6 - Settings

Jekejeke Prolog, Runtime Library 0.9.6

Configure the colors of the terminal window.

Background Color: □ Selection Color: 🟦

Input Text: 🟩 Selection Text: □

Output Text: ■

Error Text: 🟥

Background Color: The background colour of the console window.

Selection Color: The background colour of text selected by the end-user.

Selection Text: The font colour of text selected by the end-user.

Input Text: The font colour for the input stream of the console window.

Output Text: The font colour for the output stream of the console window.

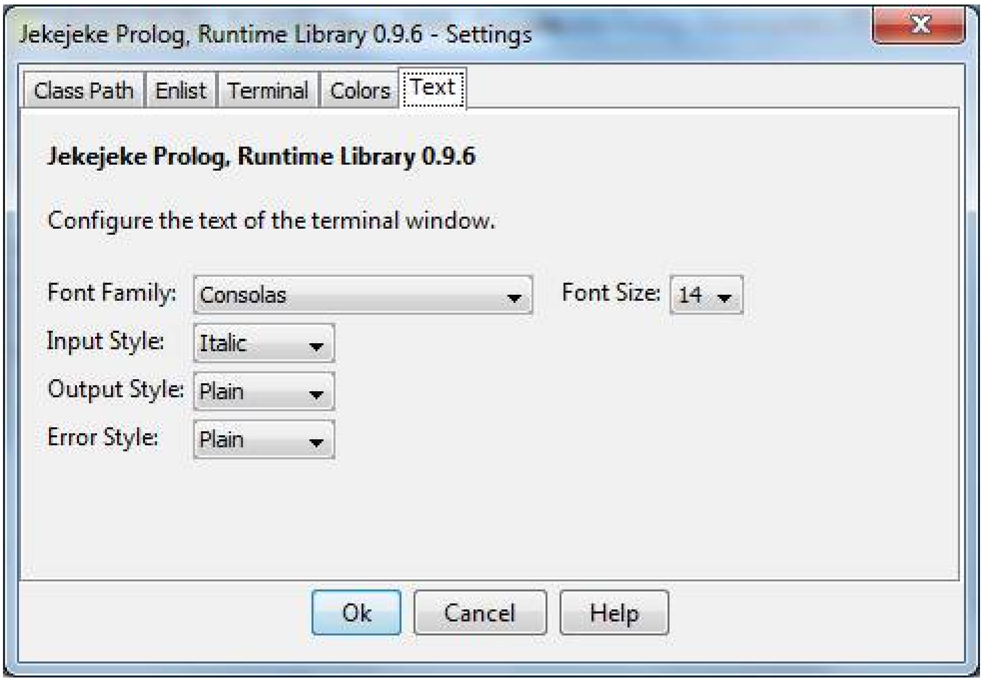
Error Text: The font colour for the error stream of the console window.

Ok: Save the settings and close the dialog.

Cancel: Close the dialog.

Help: Show this help section.

7.5 Text Panel



Jekejeke Prolog, Runtime Library 0.9.6 - Settings

Jekejeke Prolog, Runtime Library 0.9.6

Configure the text of the terminal window.

Font Family: Consolas

Font Size: 14

Input Style: Italic

Output Style: Plain

Error Style: Plain

Font Family: The font family of the console window.

Font Size: The font size of the console window.

Input Style: The font style for the input stream of the console window.

Output Style: The font style for the output stream of the console window.

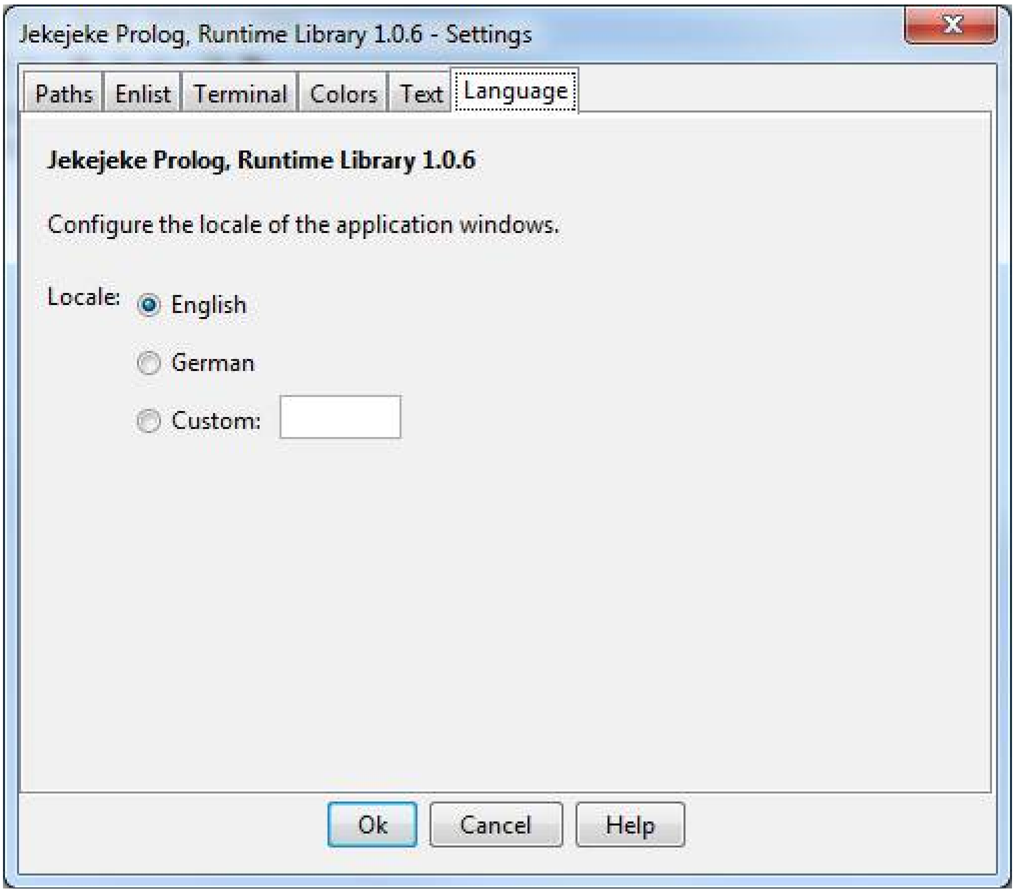
Error Style: The font style for the error stream of the console window.

Ok: Save the settings and close the dialog.

Cancel: Close the dialog.

Help: Show this help section.

7.6 Language Panel



Jekejeke Prolog, Runtime Library 1.0.6 - Settings

Jekejeke Prolog, Runtime Library 1.0.6

Configure the locale of the application windows.

Locale: ● English

○ German

○ Custom:

Locale English: The user interface locale is “en”.

Locale German: The user interface locale is “de”.

Locale Custom: The user interface locale is as specified.

Ok: Save the settings and close the dialog.

Cancel: Close the dialog.

Help: Show this help section.

8 Appendix Tour Listings

The full source code of the Java classes and Prolog texts for the tours is given. The following source code has been included:

- Text Tour
- Class Path Tour
- Threads Tour
- Memory Low Tour

8.1 Text Tour

We might insinuate that a new born greets with “Hello World!”, similarly you can check with this program that your new installation of the top-level is basically working. The following artefacts are listed in the following:

- **hello.p:** The Prolog text of the hello world example without arguments.
- **hello2.p:** The Prolog text of the hello world example with arguments.

Prolog Text hello

```
/**
 * Prolog code for the Hello World example.
 *
 * Copyright 2010, XLOG Technologies GmbH, Switzerland
 * Jekejeke Prolog 0.8.3 (a fast and small prolog interpreter)
 */

hello :- write('Hello World!'), nl.
```

Prolog Text hello2

```
/**
 * Prolog code for the hello argument example.
 *
 * Copyright 2010, XLOG Technologies GmbH, Switzerland
 * Jekejeke Prolog 0.8.3 (a fast and small prolog interpreter)
 */

hello(X) :- write('Hello '), write(X), nl.
```

8.2 Class Path Tour

With this program you can check that your Java tool chain is basically working. In a distributed environment you might want to change the text to “Phone Home”, etc... The following artefacts are listed in the following:

- **Hello.java:** The Java class without thorough exception handling.
- **Hello2.java:** The Java class with thorough exception handling.

Java Class Hello

```
import jekpro.tools.api.Interpreter;

/**
 * <p>Java code for the hello argument example.</p>
 * <p>Without thorough exception handling.</p>
 *
 * @author Copyright 2010-2013, XLOG Technologies GmbH, Switzerland
 * @version Jekejeke Prolog 0.9.0 (a fast and small prolog interpreter)
 */
public class Hello {

    /**
     * <p>Hello the argument.</p>
     *
     * @param inter The interpreter.
     * @param arg The argument.
     */
    public static void hello(Interpreter inter, String arg) {
        try {
            Writer wr = (Writer)((TermRef)inter.getProperty(
                    ToolkitLibrary.PROP_SYS_CUR_OUTPUT)).getValue();
            wr.write("Hello ");
            wr.write(arg);
            wr.write('\n');
            wr.flush();
        } catch (IOException x) {
            /* */
        }
    }

}
```

Java Class Hello2

```
import jekpro.tools.api.Interpreter;

/**
 * <p>Java code for the hello argument example.</p>
 * <p>With thorough exception handling.</p>
 *
 * @author Copyright 2011-2013, XLOG Technologies GmbH, Switzerland
 * @version Jekejeke Prolog 0.9.0 (a fast and small prolog interpreter)
 */
public class Hello {

    /**
```

```
     * <p>Hello the argument.</p>
     *
     * @param inter The interpreter.
     * @param arg The argument.
     * @throws InterpreterMessage Problems with writing.
     */
    public static void hello(Interpreter inter,
                             String arg) throws InterpreterMessage {
        Object obj = ((TermRef)inter.getProperty(
               ToolkitLibrary.PROP_SYS_CUR_OUTPUT)).getValue();
        if (!(obj instanceof Writer))
            throw new InterpreterMessage(
                    InterpreterMessage.permissionError("output",
                            "binary_stream", new TermAtom("user_output")));
        try {
            Writer wr = (Writer)obj;
            wr.write("Hello ");
            wr.write(arg);
            wr.write('\n');
            wr.flush();
        } catch (IOException x) {
            throw InterpreterMessage.mapIOException(x, inter);
        }
    }

}
```

8.3 Threads Tour

This program gives a hint how the runtime library can be used to experiment with multiple threads by means of additional windows. The following artefacts are listed in the following:

- **thread.p:** The Prolog text of the logging and the example perpetual process.

Prolog Text thread

```
/**
 * Prolog code for the multi threading tour.
 *
 * Copyright 2010, XLOG Technologies GmbH, Switzerland
 * Jekejeke Prolog 0.8.7 (a fast and small prolog interpreter)
 */

init :- current_output(X), assertz(console(X)).

log(X,Y) :-
    atom_concat('Process ',X,A1),
    atom_concat(A1,': ',A2),
    atom_concat(A2,Y,A3),
    atom_concat(A3,'\n',A4),
    console(Z),
    write(Z,A4),
    flush_output(Z).

process(X) :-
    repeat,
    log(X,'Ha'),
    log(X,'Tschi'),
    fail.
```

8.4 Memory Low Tour

This program has been used as an example in the memory low description. The following artefacts are listed in the following:

- **fac.p:** The Prolog text to compute the factorial via the Peano axioms.

Prolog Text fac

```
/**
 * Prolog code for the factorial via Peano axioms.
 *
 * Copyright 2010, XLOG Technologies GmbH, Switzerland
 * Jekejeke Prolog 0.8.7 (a fast and small prolog interpreter)
 **/

add(n,X,X).
add(s(X),Y,Z) :- add(X,s(Y),Z).

mul(n,_,n).
mul(s(X),Y,Z) :- mul(X,Y,H), add(Y,H,Z).

fac(n,s(n)).
fac(s(X),Y) :- fac(X,H), mul(s(X),H,Y).
```

Pictures

Tables

Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.

References