Frysk Demo Script

July 2006

Introduction

This is a script for demonstrating the open-source tool Frysk, which provides a combination of monitoring and debugging capabilities. The functionality that is depicted in this script should always work as of July 2006. As development on Frysk is progressing at a brisk pace, new features are being added weekly by both the Frysk development team and Red Hat customers. Every attempt will be made to keep this document current.

What is Frysk? What is it supposed to do?

The Frysk project seeks to provide developers and systems administrators with an intelligent, state-of-the-art tool that monitors the inner-workings of processes within a system (or multiple systems) and sends an alert when problems occur. Frysk "observes" processes as unobtrusively as possible and detects problems in such a way that you can interrogate the errant process for useful information before it exits the CPU queue. This information will either help resolve the problem or it will result in data that will narrow the problem down so that the next debug run can be more intelligently instrumented to harvest even more and better data.

Frysk is basically divided into two major pieces of functionality: a monitoring function where you can watch the actions of various processes through "observers" and a debugger function that you can activate either via an observer from the monitoring side of Frysk or as an independent task that has no connection to the monitoring side.

The key difference between Frysk and most open source Linux debuggers is that once an errant process has been identified, you can use Frysk to attach "observers" to it. These observers will, if the user so desires, stop the process at any sign of trouble or when a user-defined behavior is detected. In the case of an errant process, it is stopped while useful information can be retrieved via backtraces, retrieving variable values, setting breakpoints, etc. When a Frysk-observed process misbehaves, you can have Frysk perform several actions—including having e-mails sent, having warning windows appear, having a "debug" window appear with the source code, having other processes/scripts activated, and a variety of other options.

What Frysk is Not

There has been some misconception that Frysk can be used as a "profiling" tool. While Frysk can be used to "monitor" processes and the actions they perform, it does not actually keep statistics or make any performance-type calculations. There are other more suitable tools that provide this function much more efficiently and accurately, one of them being OProfile.

Observer Concept

The "observer" model Frysk uses is based on the Java "Observable" class. The theory of operation is that when an "observer" is attached to a process or a thread within a process (henceforth referred to simply as a process), it reports back to the process that initiated the observer (Frysk) that a specific behavior has occurred.

Currently the following observers can be attached to a process:

Exec – Monitor a process for a call to the "exec" function

Fork – Monitor a process for a call to the "fork" function

Task Clone – Monitor a process for a clone operation

Task Terminating – Monitor a process for any activity for it to exit the CPU queue

Syscall – Monitor a process for any system call

Custom observer – A combination of observers with possibly additional logic.

What to Run

Frysk is available in RPM format on both the Fedora 5 (and greater) and RHEL4 (and greater) operating systems. Please make sure any system that you want to install Frysk on is fully updated with all of the latest packages. Since Frysk is a state-of-the-art tool, it depends on the latest and greatest technologies and often uses the most current features of the packages it depends on for proper operation.

If you want to make sure you have the latest version of Frysk with all of the most current capabilities, you may also download/build Frysk from its CVS repository (recommended). The prerequisite packages and instructions on how to do this are on the Frysk website: http://www.sourceware.org/frysk/build. The instructions there include how to install/build on both

Fedora and RHEL4.

Demo Set Up

You need to do a couple of small things before you demo Frysk:

- First, bring up one or more bash shell windows. This will be used in defining monitoring sessions as you will attach observers to the bash process.
- Second, you need a program to demo the source window:
  - If you downloaded/built Frysk from the CVS repository, there is a demo program frysk/demo/looper.c, and a script, frysk/htdocs/demo/compile_looper.sh, to build it. This will build a small task that has debug information that will loop forever—enabling you to attach a source window to it as if you were going to debug it. Go to that directory and execute the "compile_looper.sh" script, which will create the "looper" executable that will be used later.
  - If you installed Frysk from an RPM, enter this program into a source file and store it under the name looper.c.

```
#include <stdio.h>
#include <stdlib.h>
int main(){
  printf("Main: %p\n", &main);
  printf("Pid: %d\n", getpid());
  while(1);
  return 0;
}
```

Now, compile it with this command:

```
gcc -g -o looper looper.c
```

Remember where the executable is for later in the demo.

Starting Frysk

Installed from an RPM

Frysk can be activated from the main menu under "System Tools" by clicking on the Frysk name/icon. It can also be activated from a bash shell using the following command:

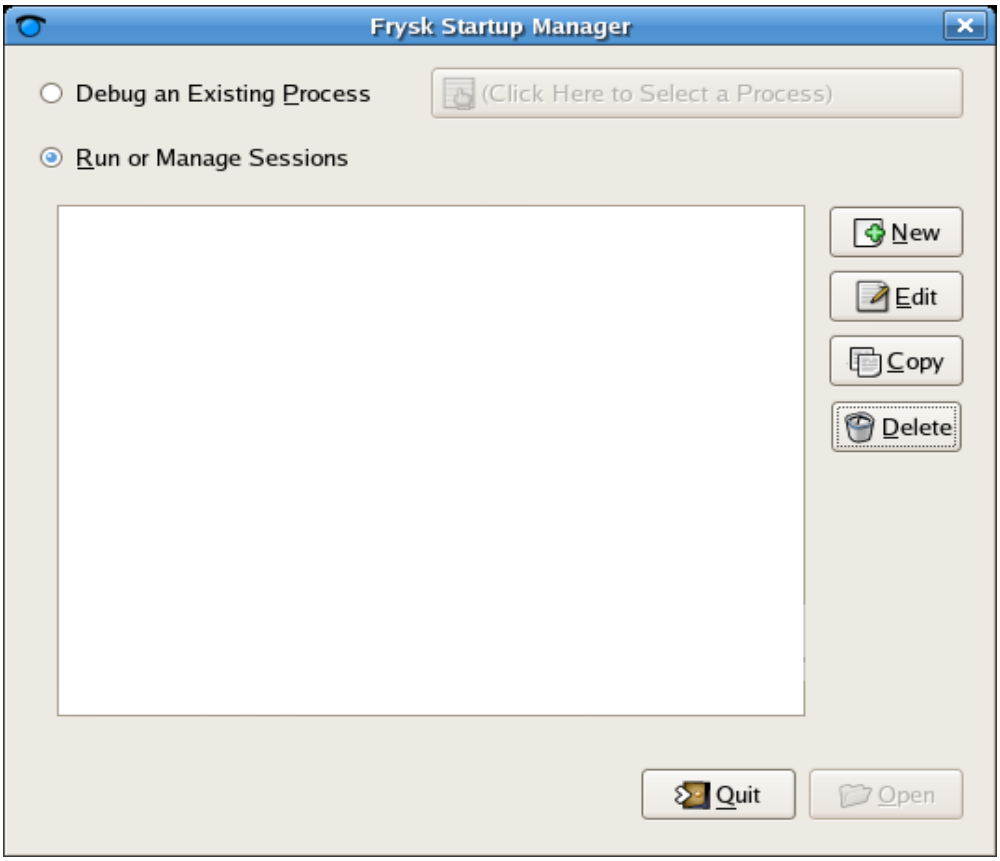
/usr/bin/frysk

Downloaded/Built

If you downloaded/built Frysk from CVS, see the instructions for activating it from the build tree under the "Run" section. (**Hint:** if you followed the instructions on the web page, the executable is in the "build/frysk-gui/frysk/gui" dir and is named FryskGui.)

Initial Frysk dialog

When Frysk is first activated, the following dialog appears:

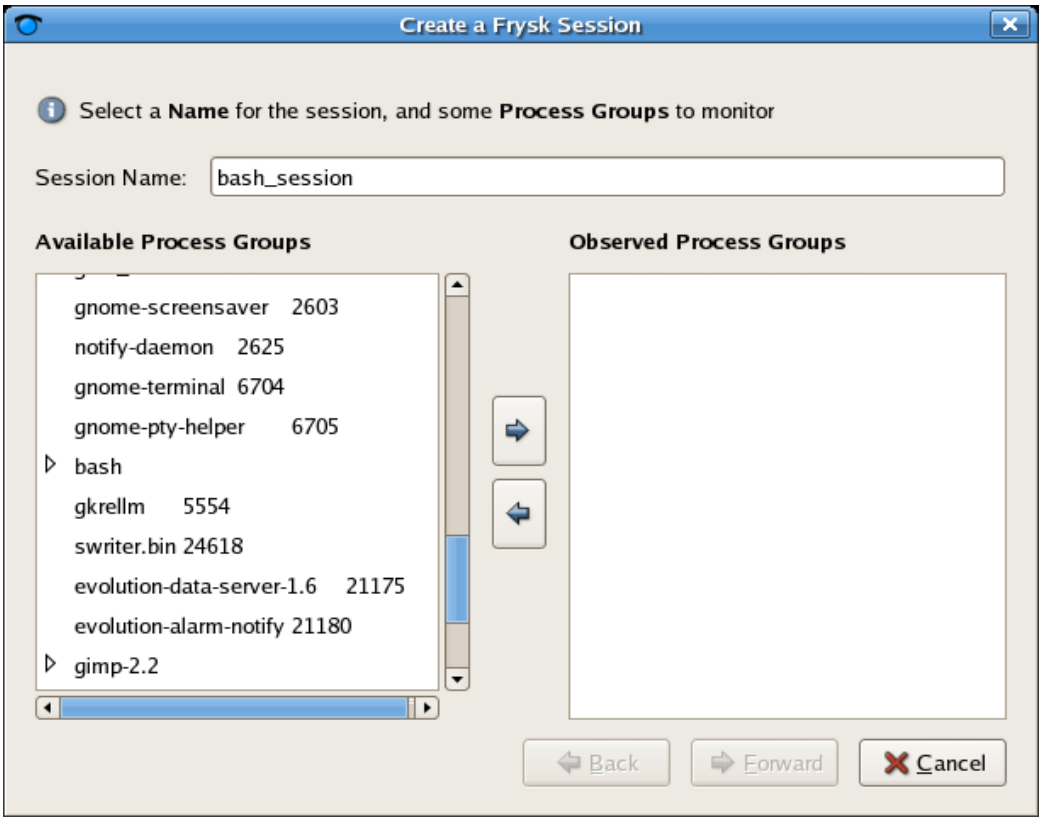


The Frysk Startup Manager window allows you to either debug a process you own and that is currently in the CPU queue, or to bring up the monitoring side of Frysk, which enables you to attach various types of observers to a process.

Selected by default is the **Run or Manage Sessions**. A *session* in Frysk terminology is a process or set of processes that you have defined as being "interesting". You can attach multiple observers to each process. Each session can have as many processes as you want with as many observers attached to them as you want. The nice thing about sessions is that once you define one, its definition is saved and it can be reused, modified, or copied for use as a basis for creating other sessions. Defined sessions remain persistent across reboots.

Create a Session

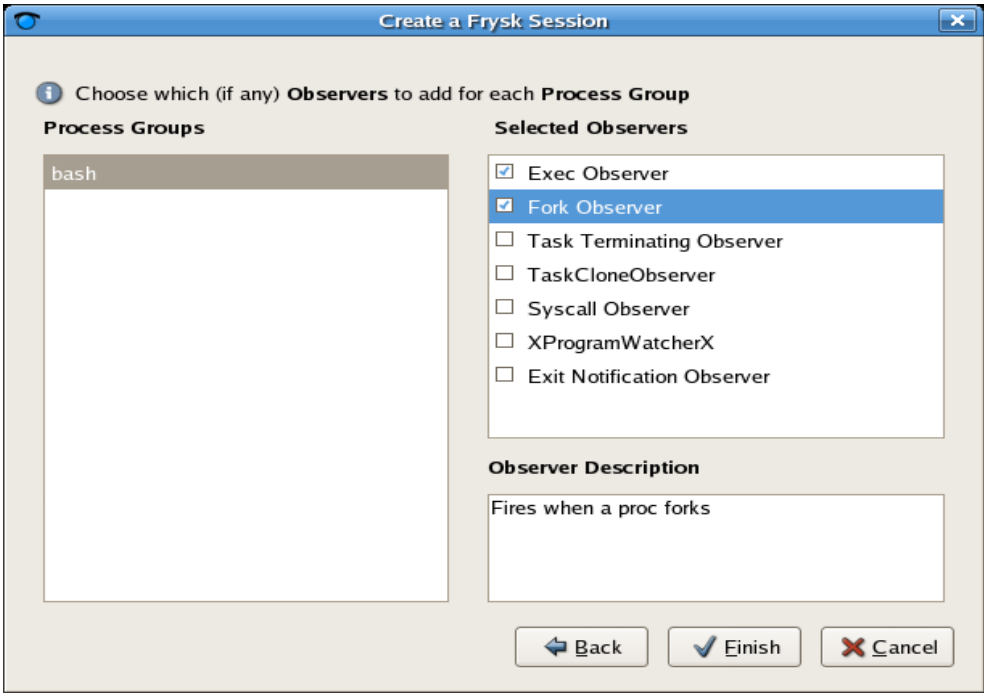
To create a session, simply click on the **New** button, which will bring up the following dialog:



In the **Create a Frysk Session** dialog you must do two things:

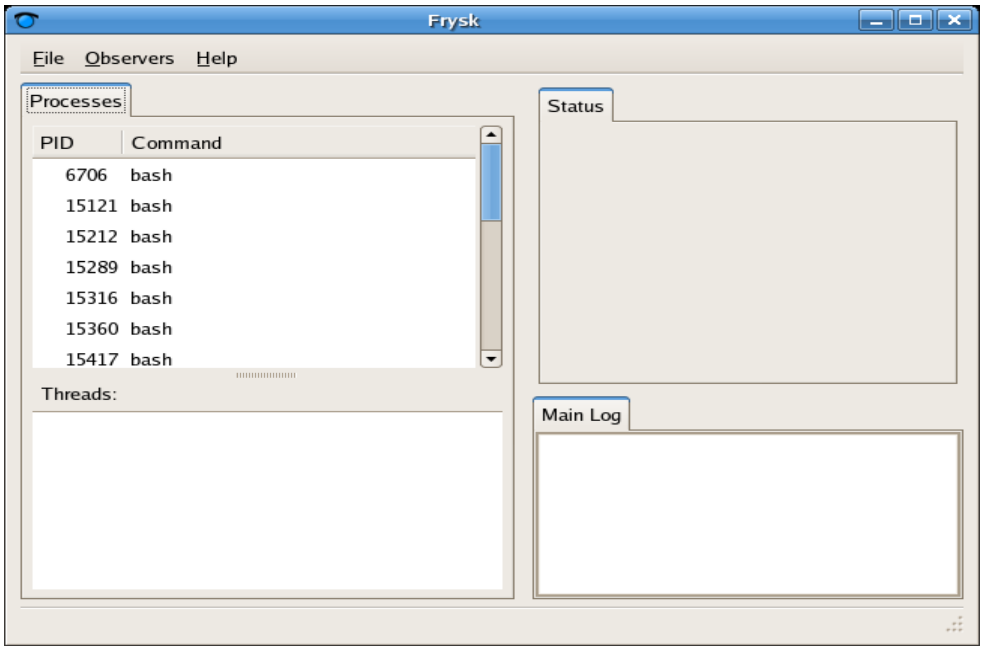
1. Select one or more processes to which you want to attach observers.
2. Name the session in the data field at the top of the dialog.

For the purposes of the demo, select "bash" from the **Available Process Groups** pane and click the "right arrow" to place it in the **Observed Process Groups** window. If there are multiple processes running with the same name, as is the case with "bash" in this example, a triangle is placed to the left of the name. Clicking on that triangle will expand the bash entry to show all of the PIDs currently in the CPU queue running under the name "bash". You could select a specific PID to monitor if you wanted, but that would not be generic enough to be persistent between Frysk executions—while just selecting bash is. Selecting just "bash" will then select all "bash" PIDs for this session and every session each time a session with this name is selected. Now type in a name for the session in the "Session Name:" field. Once these two pieces of information have been entered, the **Forward** button is enabled and you can progress to the next dialog:

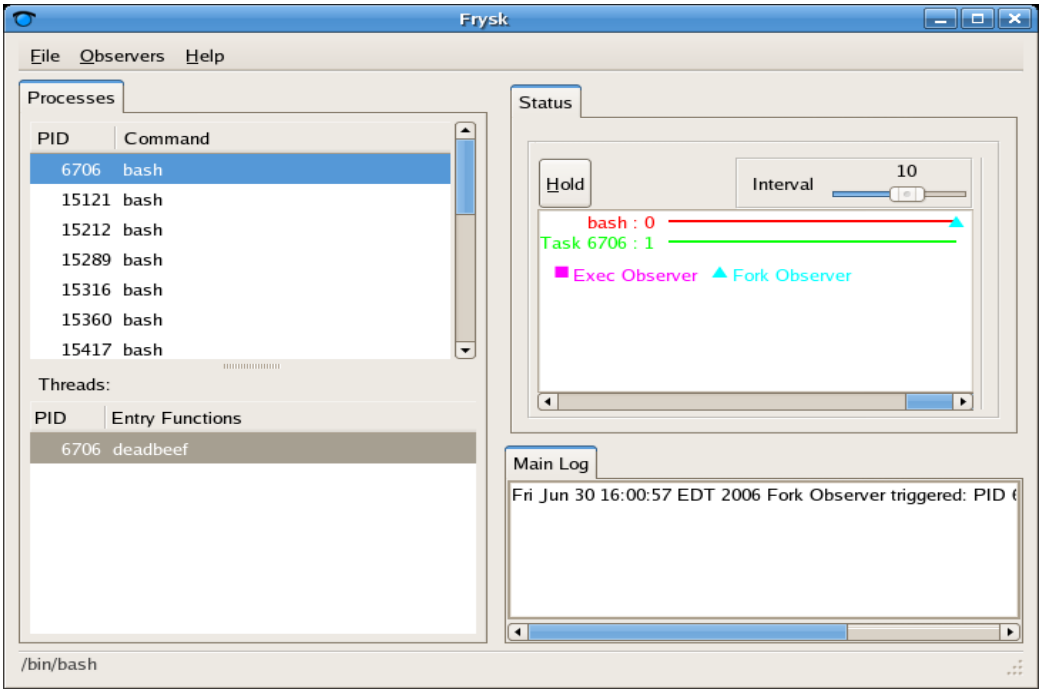


On this dialog you can select each process from the **Process Groups** pane and attach any or all of the observers from the **Selected Observers** pane. For this demo, select the **Task Terminating Observer** and the **Fork Observer** to attach to the bash process and then click the **Finish** button. You are returned to the original Frysk dialog.

From this dialog you can repeat the sequence again to define another set of processes to observe(another session) or you can move on into the monitoring mode by selecting a session and clicking on the **Open** button to start a monitoring session. For this demo, select the session with the bash you just defined and open it to bring up something similar to the following:



As can be seen, there were several bash processes running when this session was started. Each of those processes now has both a “Task Terminating Observer” and a ”Fork Observer” attached to it. Now click on any of the processes in the **Processes** pane and the dialog will transform into the following:



All four quadrants of the dialog are now active. Here is a description of each:

Processes: Shows all processes defined for this session

Threads: Shows the threads of the process selected in the **Processes** pane

Status: Shows a timeline graph of each observer attached to the process selected in the **Processes** pane and when the observers fired (a different color for every observer)

Main Log: Shows a text-based history of the observers that have fired for all of the processes defined in the session.

Now, if there are multiple bash processes running, go to the bash terminal window previously opened and do an "echo $$" to get its PID and select that PID in the **Processes** pane.

To see activity in the **Status** pane, enter the following small bash script in the bash terminal window:

```
for i in `ls -1 /etc`; do ls -ls /etc/$i; done
```

This should result in several of the "Fork Observer" symbols being shown in the status view. A symbol is shown each time a fork is made.

Adding Observers

If you want additional observers attached to any process, simply right-click on the process in the **Processes** pane and a menu will appear. Select the **Add observer** item from that menu and a list of attachable observers will appear. Simply select the observer you want attached to the selected process.

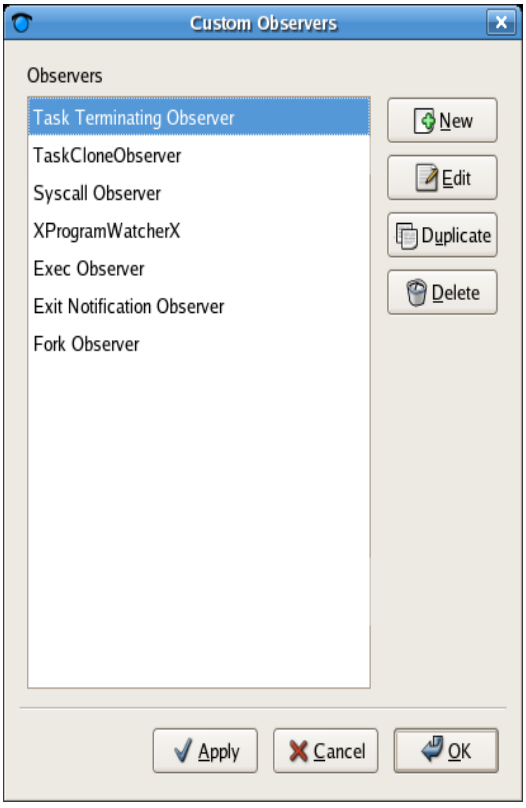
What is a Custom Observer?

Now, suppose there is a need to do something other than "monitor" a process and plot/log its activities. Say, for instance, you want to stop a process, activate a source window to look at the source, look at variable values and/or debug it when an observer fires.

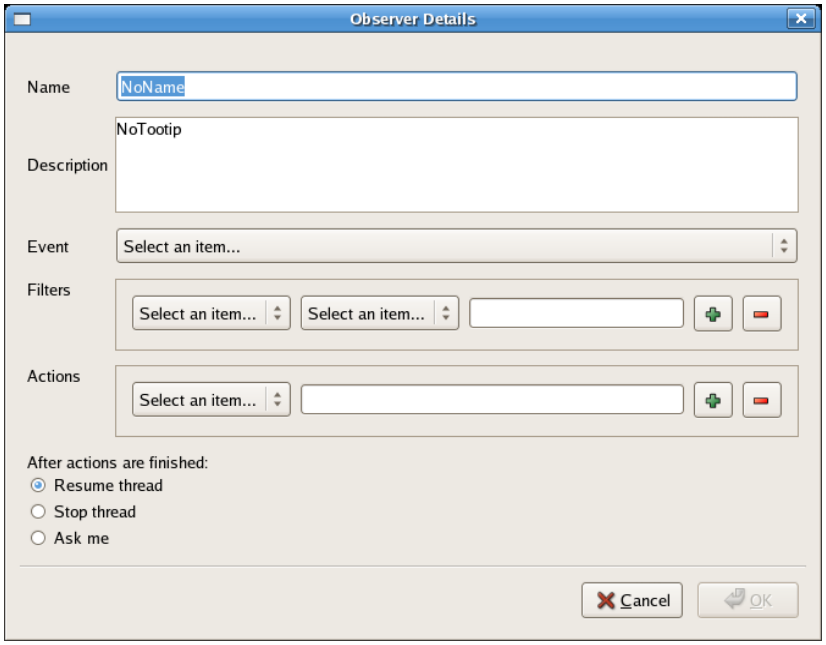
Another major feature that custom observers provide is the ability to "filter" events. For example, suppose a user wants to monitor process and fire an "exec" observer only when it tries to "exec" a process foo. Or, maybe fire an "exec" observer when it tries to "exec" any process other than foo. The custom observer feature has been provided for this type of circumstance and many others

Creating a Custom Observer

The creation of custom observers begins with the **Monitoring Window**. Select the **Observers** item from the **Monitoring** window and then select **Manage Custom Observers...** to get a window that looks like this.



From this window the user can modify one of the existing pre-defined observers or create a new observer based on one of the existing observers. Usually it is best to create a new observer if the user decides to apply filtering. Click the **New** button to create a new custom observer with the following window.



By filling out the fields in this window the user can create an extremely flexible observer that can perform many functions. Here are the fields on this display:

Name: Name of the custom observer

Description: Description of the observer

Event: Which system observer you want this observer based on (Exec, Fork, Task Terminating, TaskClone, SysCall)

Filters: The values for this field are different depending on which observer has been selected. For example, if the Fork observer was selected in Event: the user can select either "Name forking thread" or "Name forked thread" for the first **Select an item...** field. The second **Select an item...** field will then allow the user to choose "is" or "is not". The data-enterable field allows the user to specify the name of the task that will trigger the observer. The **+** button allows the user to add more filters while the **–** button allows the user to delete previously-defined filters.

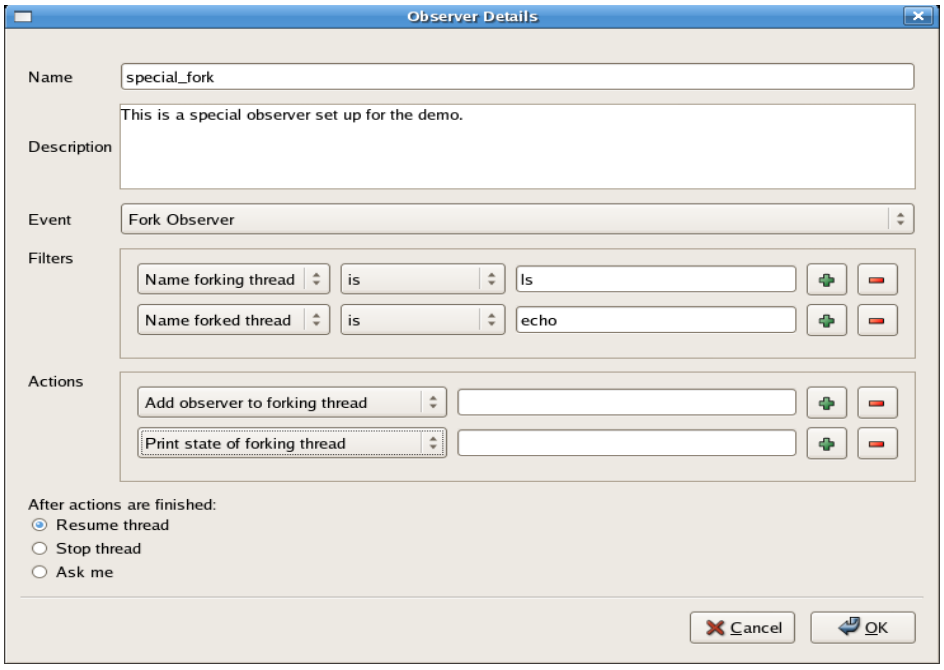
Actions: This field is similar to Filters in that it varies depending on which observer is being used for this observer. The actions that can be specified for the **Fork** observer are:

- Log event
- Show source code of forking thread
- Add observer to forking thread
- Print state of forking thread
- Show source code of forked thread
- Add observer to forked thread
- Print state of forked thread

The **+/–** fields perform the same functions as the Filters +/- by allowing the user to add multiple actions or delete previously-defined actions.

After actions are finished: These radio buttons tell Frysk what to do after it has completed the actions specified from the "Actions" field.

Below is a screenshot of a custom observer with 2 filters and 2 actions defined for it.



Source Window

Current Capabilities

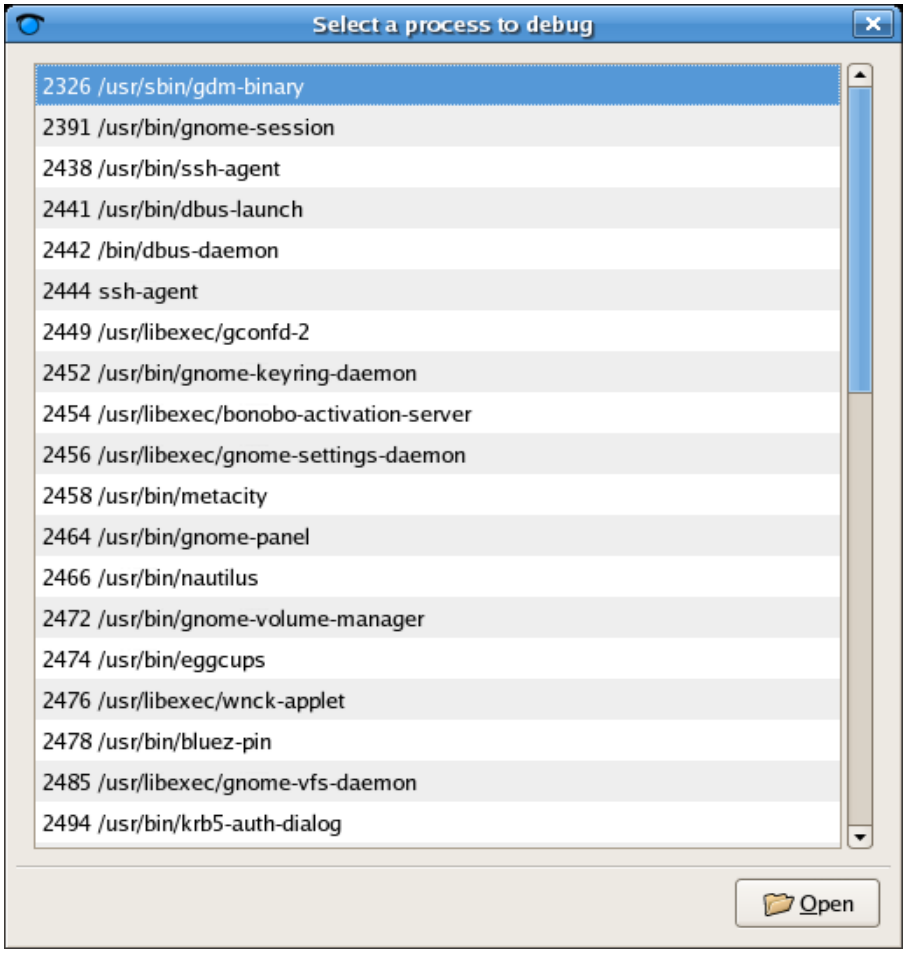
The source window is still in its early stages of development so it currently has limited capabilities. In fact, only just recently has the source window been brought out of demo mode and into the mode where the source code from actual processes can be viewed. Thanks to new resources being allocated to Frysk in the last few weeks, steady progress is being made and new features are being added almost daily, so keep checking back.

Setup

Before activating the source window, a process needs to be created that has debug information associated with it. The previously compiled looper program should be activated from a shell at this time. It will print out its PID and the address of where the main program starts. This process is made to stay in the CPU queue indefinitely until aborted by you.

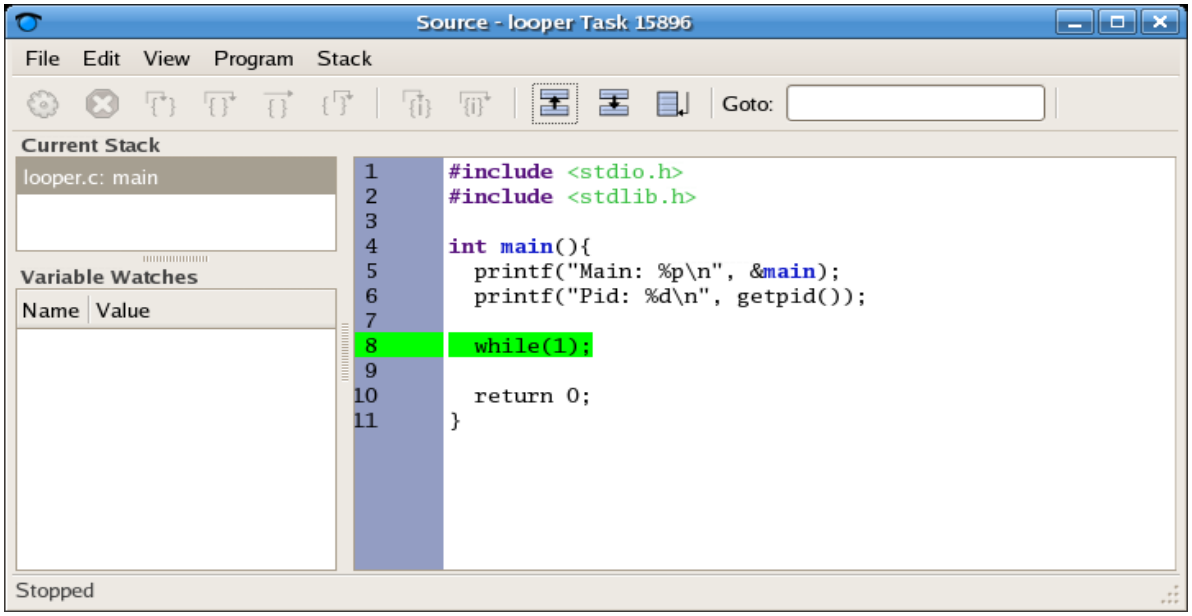
Activation

The source window can be activated from the initial Frysk Startup Manager dialog by first selecting the **Debug an Existing Process** radio button. The field to the right (**Click Here to Select a Process**) will become active. By clicking on that field you will be presented a list of processes that you own—as shown here:



To activate the source window, either select a process from the window and click **Open** or double-click on the process. If you select a process that does not have debuginfo associated with it, (compiled with the "-g" option), a message will appear stating that fact.

To bring up a source window that *has* debuginfo, select the "looper" process from the list and open it. A source window with the code of the looper program appears:



This window contains the source code of the module selected in the **Current Stack** window. The source window has three panes: **Current Stack**, **Variable Watches**, and the source itself. The **Current Stack** and **Variable Watches** are not implemented right now, but watch for them soon. Also, notice the grayed out icons on the toolbar—those items are currently unavailable, but activating that functionality is a high priority for the Frysk development team.

Notice that Frysk does the usual things that GUI-based debuggers do:

- key word highlighting
- variable highlighting
- executable lines marked by a dash ("-").

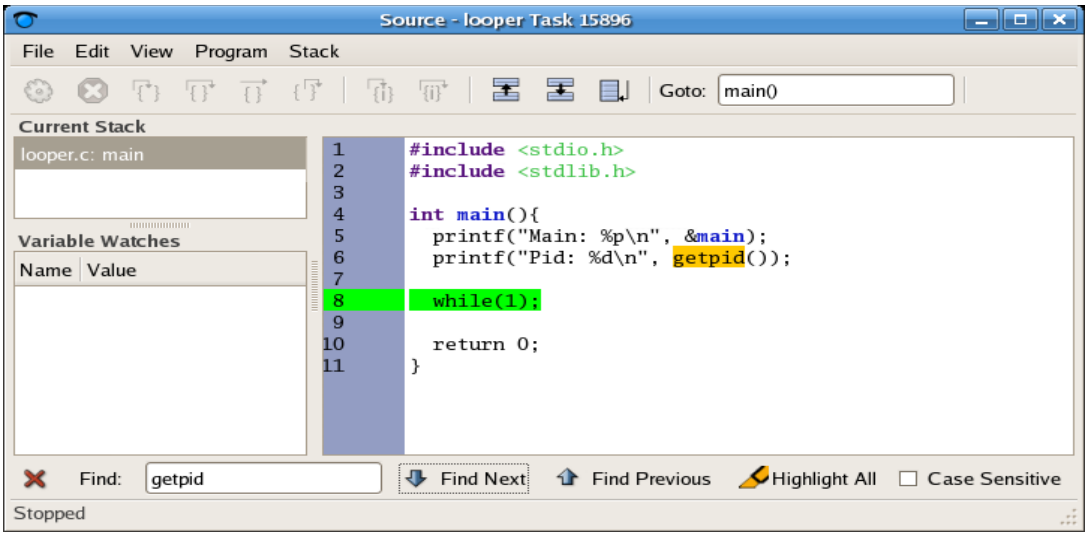
You can select the colors and types (bold or italic) of highlighting by clicking **Edit > Preferences**.

"Goto Function" feature

Notice that line number 8 is highlighted. That is where the current line that is being executed. Also, the "Goto:" field enables you to go to various functions in the code. Click in the field and type the letter "m". Notice that a pop-up appears for auto completion for "main()". Selecting "main()" will position the source window so that it is in the center of the dialog.

"Find" function

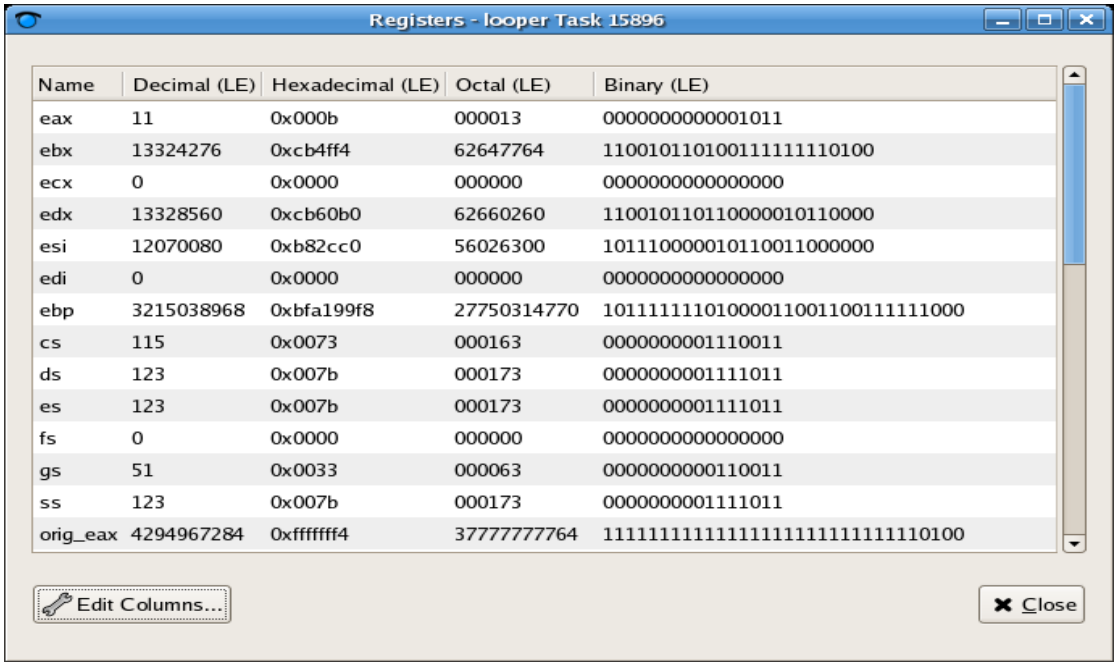
To activate the **Find** feature in Frysk, click **Edit > Find**. The bottom of the window expands and looks similar to the "Find" feature for Mozilla Firefox.



The **Find** feature will find any character string entered into the field while the **Goto** feature is designed to find only functions. In the above screenshot "getpid" has been entered to be found and **Find Next** clicked. As a result, "getpid" has been highlighted.

Register Window/Memory Window

To view the current values of all of the registers for the current task, simply click **View > Register Window**. A window similar to the following appears:

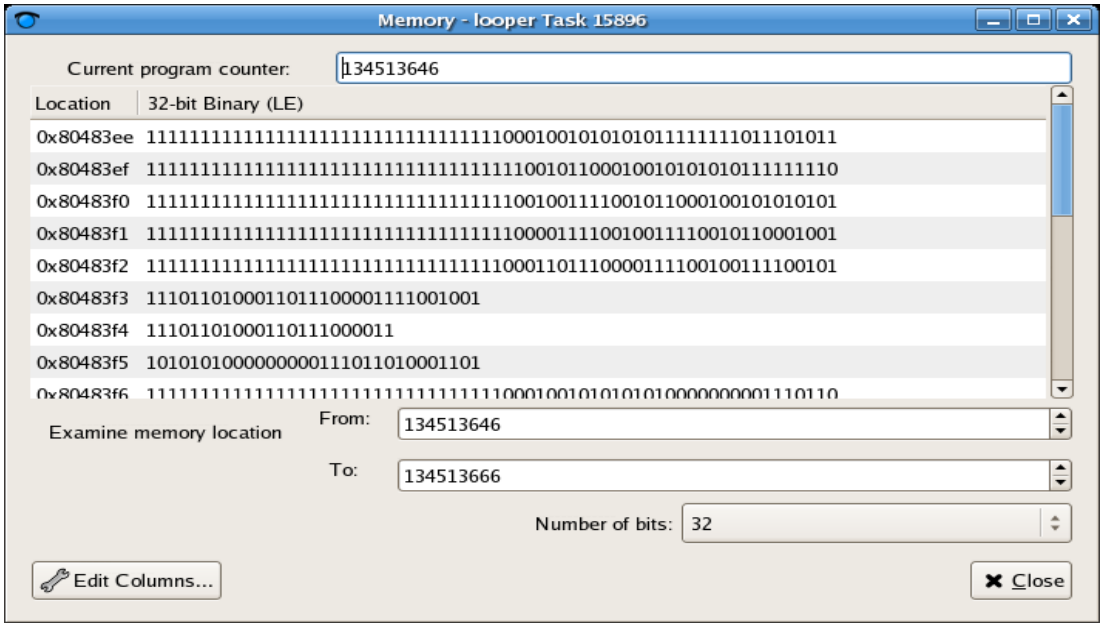


The **Registers** window you just called up may not look quite like the above, but you can get it there by clicking on the **Edit Columns...** button. By clicking that button, a window is provided that enables you to select the various formats of how you want the data displayed. You can select as many or as few formats as desired.

Another feature of the **Registers** window is that the column order of the formats from left to right can be changed using the "drag-and-drop" method. To move a column, simply left-click on the column heading and hold it and drag it to the new position and release it. The column will settle into its new position.

Memory Window

To activate the **Memory** window, click **View > Memory Window**. A window similar to the following appears:



This window is very similar to the **Registers** window in capabilities in that you can vary the format of the data being displayed via the **Edit Columns...** button. Also, the range of memory locations being viewed can be varied with the **From**: and **To**: fields at the bottom and the register width can be varied by the **Number of bits:** field.

Current Stack View (future)

This is the list of source modules that are associated with the selected PID/thread. Each item in the list is selectable and when it is, its source code is displayed in the source window view.

Variable Traces View (future)

This view (when it becomes functional) will contain a list of variables for which you have chosen to follow the values.

Roadmap for Frysk Source Window

The following features are currently planned for the Frysk source window:

(Note this is just a few of the major enhancements planned and is not meant to be an all-inclusive list).

- Display variable values when hovering over the variable
- Display next Program Counter(PC)
- Highlight optimized code
- Corefile debugging
- Macro expansion
- Code folding/expansion
- Column breaks
- Multiple source window tabs
- Stop at any/all throw of exceptions