
Using Glib Signals with GOB

Ikke

Version 0.1

Copyright © 2005 Ikke

Published under the "Creative Commons Attribution-ShareAlike License Belgium"

Table of Contents

Introduction	1
Our first GOB file	1
The C part	2
Build time	3
What comes next?	3

Introduction

I'd like to take you on a little "Glib [<http://developer.gnome.org/doc/API/2.0/glib/>]/GObject [<http://developer.gnome.org/doc/API/2.0/gobject/>] Signals [<http://developer.gnome.org/doc/API/2.0/gobject/gobject-Signals.html>]" tour today.

First of all, what are Signals? The GObject Signal API [<http://developer.gnome.org/doc/API/2.0/gobject/gobject-Signals.html>] states: "Signals - A means for customization of object behavior and a general purpose notification mechanism"

Signals can be compared to .Net Delegates/Events, C/C++ callbacks using function pointers, or Java ActionListeners (at some extend).

Today we'll create some fancy (well ;-)) GObject in C which shows us some basic usage of signals and signal handling.

Our first GOB file

First, we need to create a simple GOB file, like this:

```
requires 2.0.0
```

```
class Test:Signal from G:Object
{
    //signal signature
    signal last NONE (NONE)
    void
    testsignal (self)
    {
        g_print ("Signal broadcast requested\n");
    }

    public void
    sendsignal (self)
    {
        //Call our signal function. Functions attached to this signal w
        self_testsignal (self);
    }
}
```

```
public GObject *
new (void)
{
    g_print ("New TestSignal requested\n");
    return (GObject *) GET_NEW;
}
}
```

"requires 2.0.0" tells GOB we want at least version 2.0.0 of GOB to parse our file.

Then we define a new class, Test:Signal (Test is the namespace, Signal is the classname) which derives from G:Object (GObject).

Now we declare a simple signal, which takes no arguments ('cause we like to keep it simple, isn't it?). If the signal would take arguments we could use them in the handler, here we only output some debug message.

Next comes a simple public function of our object (and yes, this is plain C) which in turn calls our signal emitter. Notice the use of "self_testsignal" here. The "self_" part will be nicely converted by GOB to the correct function name. Like this we can rename our object (make it "Test:SignalTest" or something) without breaking the code in the GOB file itself. A lot of C preprocessing macro's are generated to achieve this functionality, read the generated .c file (see next step) to see how it's done.

Last but not least we define a simple "new" function to make our TestSignal object more usable.

The C part

Ok, so now we got our GObject definition, time to generate some C code out of it. We defined an object called TestSignal, which will be declared in test-signal.[c|h], so it's quite common to call our GOB file test-signal.gob.

We let GOB do its work:

```
gob2 test-signal.gob
```

This should generate no output to stdout.

In your working directory, there should be 2 new files: test-signal.c and test-signal.h. Don't look at the .c file too much, it is not meant to be very readable. The .h file should be easy to read though.

Next step is creating some C code to test out our object and its signal emission.

```
#include "test-signal.h"
#include <glib.h>

//We will attach this function to the TestSignal's testsignal signal
void signalhandler(TestSignal *self, void *data) {
    g_print("In signal handler\n");
}

int main()
{
    TestSignal *to;

    //Initialize GObject framework
    g_type_init();

    //Create a new TestSignal object
    to = (TestSignal *) test_signal_new();

    //Attach the signal handler
    //Watch out: double underscore!!!
    test_signal_connect__testsignal (to, signalhandler, NULL);
}
```

```
    //Call some function we made
    g_print("Requesting to emit signal...\n");
    test_signal_sendsignal (to);
    g_print("Done\n");

    //Destroy the object
    g_object_unref (to);

    return 0;
}
```

I hope the code comments makes everything easy to read and understand...

Build time

Last step is compiling our code, and linking it with the necessary libraries. A quick and dirty command to do this (of course we write a Makefile for this, don't we? Ok I'll explain how to do that later ;-))

```
gcc -Wall `pkg-config --libs --cflags gobject-2.0 glib-2.0` -o testsignal test-
```

If that gives no nasty compilation/linking errors, we're set. Try running the program:

```
ikke@moonwalker ~/code/tests $ ./testsignal
New TestSignal requested
Requesting to emit signal...
In signal handler
Signal broadcast requested
Done
```

Which is what we expected. One thing to notice though: our signal is sent before the code in our testsignal function is executed.

What comes next?

That's it. Nice stuff, isn't it? And simple as hell to write :-) GObject are very very powerful things. Not only in C: if you write a GObject in C, it's very easy to create a Python wrapper [<http://www-106.ibm.com/developerworks/linux/library/l-wrap/>] for it, etc.

That's it for now, I'll try to write more (useful?) stuff if I explore new parts of GObject coding :-)