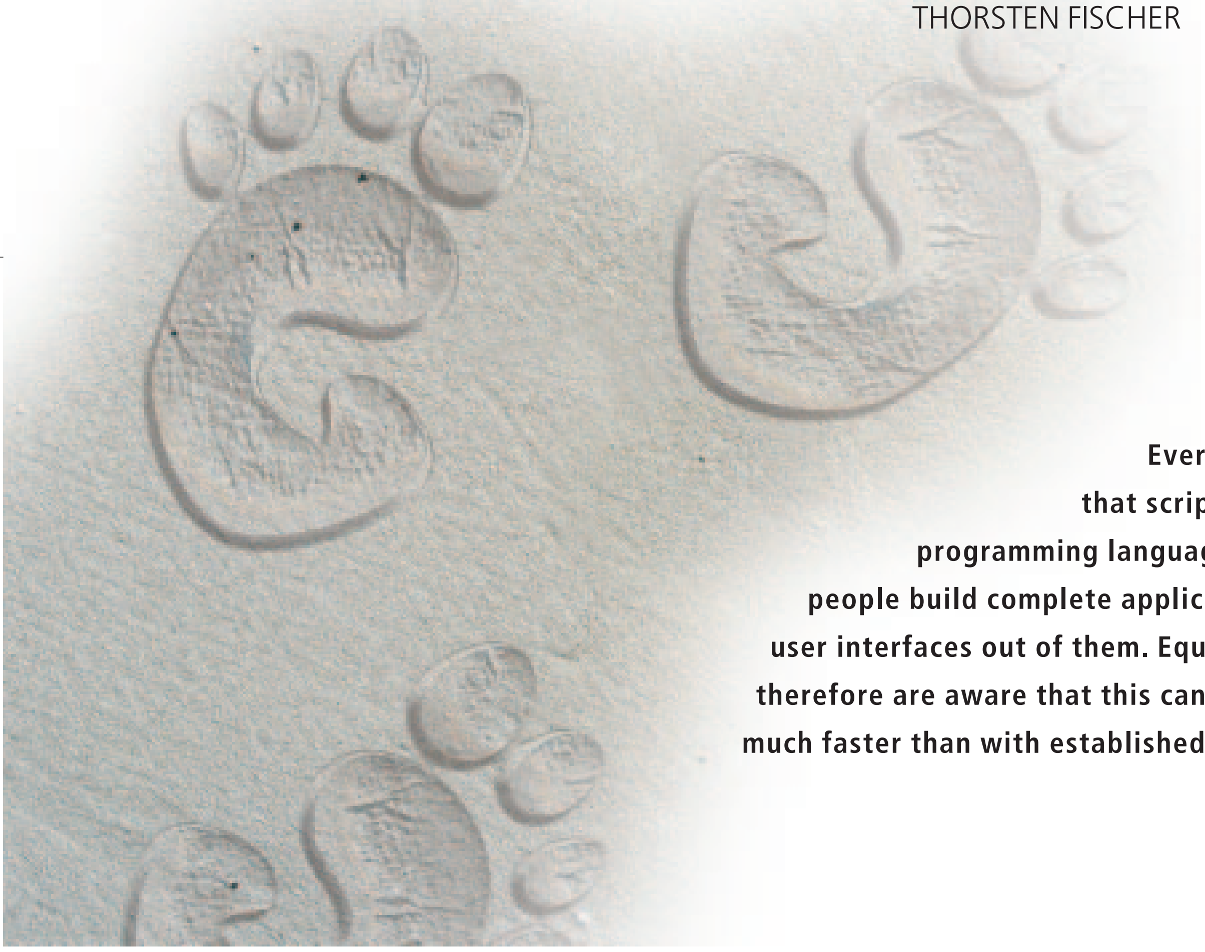
GNOME and Perl

PEARLS BEFORE DWARVES

THORSTEN FISCHER



Everyone knows by now that script languages are full programming languages. But only a few people build complete applications with graphical user interfaces out of them. Equally few people therefore are aware that this can be done much faster than with established languages.

One of the greatest strengths of Perl is the *CPAN*, the *Comprehensive Perl Archive Network*. This world-wide network was established to give Perl programmers access to the truly vast number of modules for their language. As well as small libraries and useful trivia there are also wrappers around fully developed tools for programming graphical user interfaces. Of course Gtk+ and GNOME have to be in on the act, and hence there is *Gtk-Perl* by Kenneth Albanowski. You can get your hands on the necessary module with the simple installation procedure, to which one has become accustomed by Perl via the CPAN, as follows:

```
# perl -MCPAN -e shell
cpan shell – CPAN exploration and modules ↵
installation (v1.59)
ReadLine support enabled
cpan> install Gtk-Perl
```

Gtk-Perl is presently available in version 0.7004 and contains support for GNOME. The version of CPAN is, in the circumstances, not the latest; which is why it is worth taking a look at the Gtk-Perl Homepage and seeing if there is anything new.

First steps

Normally, I prefer Python, for running up little Gtk+- or GNOME programs, as Perl is set aside on my system for administrative tasks, and therefore it seems a good idea to start with a little *Hello frog* program to get used to the language. Listing 1 shows an example for such a program. The associated screenshot can be seen in Figure 1.

Listing 1: hellofrog.pl

```
1: #!/usr/bin/perl -w
2:
3: use strict;
4: use Gnome;
5:
6: my $APPNAME = 'Hello froggy!';
7:
8: init Gnome $APPNAME;
9:
10: my $app = new Gnome::App $APPNAME, $APPN➘
AME;
11:
12: my $button = new Gtk::Button "Hello frog➘
gy!";
13: $app -> set contents ($button);
14:
15: show all $app;
16:
17: main Gtk;
```

So far, no surprises for either Perl or GNOME programmers. The module is integrated in line 4 and the application is then initialised in line 8. The application is an in-house widget in GNOME which on the other hand takes on diverse additional tasks. So for example it is easy for several instances to create one and the same application. This is reflected in line 10 in the creation of this widget. In the same way, a button is created and inserted into the window and then displays the entire application. In line 17 the program loop *gtk_main* gets control. Up to now this looks fairly similar to a Hello World in C, with the difference that Perl again exercises its magic, to demand considerably fewer keystrokes from the programmer. Incidentally, if a more alert reader who has already browsed through the material wonders why the name of the program does not appear in the title line of the screenshot, although it ought to be placed there by line 10: this is due to my misconceived Sawfish theme.

Signals and events

Gtk+ is an event-based widget set, and events certainly get the worst of it here. If an event occurs, a corresponding signal is emitted, to which a function defined by the programmer can react. Such a function is known as a callback. If the program in Listing 1 is to be ended, for example the insertion of the following code between lines 12 and 13 is standard:

```
signal connect $button 'clicked', sub { Gtk -> main quit;
                                        print "Bye-bye froggy!\n";
                                        return 0; };
```

If the button as so the control, sends out the signal 'clicked', the said anonymours function should be executed, which then leads to the ending of the program. Take note that the following cannot function:

```
signal connect $button 'clicked', Gtk -> mai➘
n_quit;
```

It's good form to write individual callbacks for each signal, especially for those which end the program, since this leaves options for clearing up – for example one can construct a routine *end* and reference it as follows:

```
sub end {
   print "Bye-bye froggy!\n";
   Gtk -> main quit;
   return 0;
}
# ...
signal_connect $button 'clicked', \
```

Getting dolled up

In a GNOME application, three things form part of the standard equipment: a status bar at the bottom end of the program, a menu at the top edge and under this a toolbar, providing access to the most frequently used functions. With Perl these things can be added quickly and easily, as can be seen from Listing 2, which took only a short time to develop; admittedly, with a little cut & paste, but that's what GPL code is there for, after all.
This code is already a great deal larger, but it also achieves a great deal more. Firstly of course the initialisation, and this time I am also giving it a version number. In lines 10, 16 and 23 the callbacks are defined, and I will go into this more later. In line 33 and the following lines the toolbar is placed. A list of lists has to be given to the function; but since there is no such thing as two-dimensional arrays in Perl, one has to make do with the corresponding notation. The reader who is so inclined should again watch for the invocation in line 47, in which the callback is defined in case the respective button on the toolbar is pressed. In the example I have used only so-called 'Stock Pixmaps', which are pre-installed in GNOME. The menus are created from the same pattern. Line

Figure 1: Hello froggy! Hello Hello froggy!

80 refers to the first interesting callback: The About-box, which can also be seen in Figure 3. The callback invocation brings the information dialog onto the screen – well, it’s not really a dialog, just an *OK* button, but anyway. Line 86 shows an element which ought to be included in every GNOME application: a

Listing 2: bookman.pl

```
 1: #!/usr/bin/perl -w
 2:
 3: use strict;
 4: use Gnome;
 5:
 6: my $APPNAME = ‘Book manager’;
 7: my $APPVERSION = ‘0.1.0’;

 8: init Gnome $APPNAME;
 9:
10: sub end {
11:   Gtk -> main quit;
12:   parint "Bye bye.!\n";
13:   return 0;
14: }
15:
16: sub infobox {
17:   my $about = new Gnome::About $APPNAME, ➘
$APPVERSION,
18:     ‘(c) 2000 Thorsten Fischer’, [‘Thors➘
ten Fischer@mapmedia.de>’],
19:     ‘Gtk-Perl sample code for Linux Magaz➘
ine.’;
20:   show $about;
21: }
22:
23: sub select {
24:   my ($clist, $row, $column, $event, @dat➘
a) = @ ;
25:   my $text = $clist -> get text ($row, $c➘
olumn);
26:   print "The selection was made in line $➘
row, column $column.\n";
27:   print "Content: $text\n";
28: }
29:
30: my $app = new Gnome::App $APPNAME, $APP➘
NAME;
31: signal connect $app ‘delete event’, \
32:
33: $app -> create toolbar (
34:   {
35:     type => ‘item’,
36:     label => ‘Open’,
37:     pixmap type => ‘stock’,
38:     pixmap_info => ‘Open’,
39:     hint => ‘Open book list’,
40:   },
41:   {
42:    type => ‘item’,
43:    label => ‘Exit’,
44:    pixmap type => ‘stock’,
45:    pixmap info => ‘Quit’,
46:    hint => "Quit $APPNAME",
47:    callback => \
48:   }
49: );
50:
51: $app->create menus (
52:  {
53:   type => ‘subtree’,
54:   label => ‘ File’,
55:   subtree => [
56:    {
57:     type => ‘item’,
58:     label => ‘ New’,
59:     pixmap_type => ‘stock’,
60:     pixmap info => ‘Menu New’
61:    },
62:    {
63:     type => ‘item’,
64:     label => ‘ Quit’,
65:     pixmap type => ‘stock’,
66:     pixmap info => ‘Menu Quit’,
67:     callback => \
68:    }
69:   ]
70:  },
71:  {
72:   type   => ‘subtree’,
73:   label  => ‘ Help’,
74:   subtree => [
75:    {
76:     type => ‘item’,
77:     label => ‘ About...’,
78:     pixmap type => ‘stock’,
79:     pixmap info => ‘Menu About’,
80:     callback => \
81:    }
82:   ]
83:  }
84: );
85:
96:
87: $appbar -> set status (‘Welcome!’);
88: $app -> set statusbar ($appbar);
89:
90: my $sw = new Gtk::ScrolledWindow undef, u➘
ndef;
91: $sw -> set policy (‘automatic’, ‘always’);
92:
93: my @list title = (‘Author’, ‘Title’, ‘IS➘
BN’);
94: my $liste = new with titles Gtk::CList (@➘
list title);
95: $liste -> signal connect (‘select row’, ➘
\);
96:
97: my @book1 = (‘Larry Wall, et al’, ‘progr➘
amming Perl’, 1565921496);
98: my @book2 = (‘Helmut Herold’, ‘Linux Unix ➘
System programming’, 3827315123);
99: my @book3 = (‘Wiglaf Droste, Gerhard Hens➘
chel’, ‘The Mullah from Bullerbü’, 389401352➘
24);
100: $liste -> append (@book1);
101: $liste -> append (@book2);
102: $liste -> append (@book3);
103:
104: for (my $i = 0; $i < $liste -> n columns; ➘
 $i++) {
105:   $list -> set column width ($i, $list -> ➘
optimal column width ($i) + 5);
106: }
107:
108: $sw -> add ($list);
109:
110: $app -> set contents ($sw);
111: $app -> set default size (640, 480);
112: show all $app;
113:
114: main Gtk;
```

status bar, which consists under GNOME of an actual status bar for messages and a progress bar which can display the progress of actions. Since in the example, both are meant to be present, the necessary flags are set to 1 instead of 0. The second parameter is the 'Level', at which the interaction takes place. In this case it is at a user-defined setting. These settings can be adjusted in the GNOME control centre.

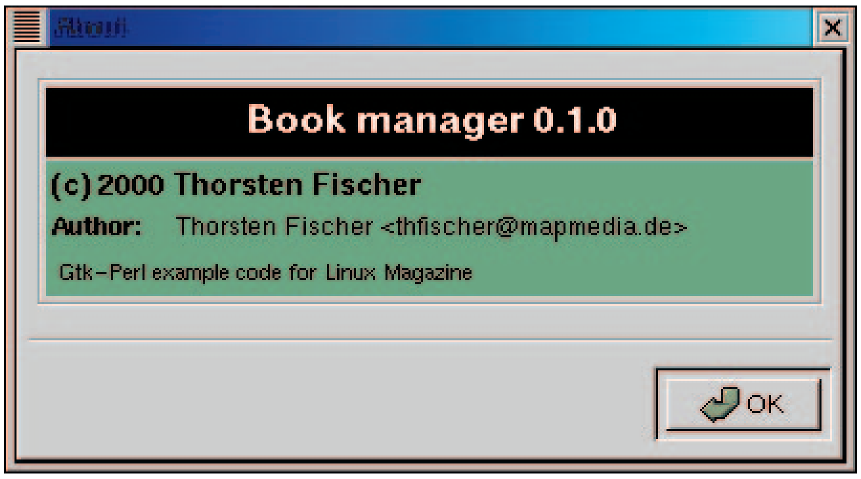
Callbacks make you feel at home in C

A GtkScrolledWindow is now stuffed into the application, into which a GtkCList with illustrious contents migrates: From line 97 the list is filled with data, in this case with a few very nice books. Messrs. Droste and Henschel will surely be pleased at being included with Wall and Herold in a 'List of nice books'. In line 104 a small loop runs, which calculates the best size for displaying the field contents. After this the whole mess is actually displayed and the program started. The callback in line 23 for selection in the list writes the data transferred in its own local variables. The structure of the callbacks for individual signals follows those in C, so that in Perl, too, one is forced to know these precisely, so as to be able to distribute the values correctly. The relevant literature does however provide references. Figure 2 now shows the completed program. 'Book manager' is perhaps a bit too high-faluting for a simple GtkCList, which more or less randomly contains a list of books, but it is only meant to be an example.

Data on the list

It is unlikely that one would want to prepare his data so that it fits into the list, only then to have to again tediously cobble it together for the rituals of a selection. For example: If I have an object in Perl having the properties of a book, then I would like to see this object associated with the entry in the line, without having to expend any effort on having to reconstruct the object purely from the entries. In order to realise this, lines with data can be connected. This is done as follows:

```
$list -> set_row_data (0, $object);
```



Book manager 0.1.0

(c) 2000 Thorsten Fischer

Author: Thorsten Fischer <thfischer@mapmedia.de>

Gtk-Perl example code for Linux Magazine

Figure 3: Information about the book manager.

Here the first line assigns the object *$object*. This assignment is retained, even if the position of the line changes as the result of a sorting. Through a variant of this function, if the line has been destroyed – perhaps it has just been deleted – a callback function can be invoked which is given the data, and thus can then proceed as required:

```
$list -> set row data full (0, $object, ➘
&function);
```

Conclusion

The nice thing about a GNOME program in Perl is that unlike C, no complete code tree has to be delivered. Basically a single file is sufficient, which contains the script or the program respectively. The code trees, as used for C programs, mainly serve to configure the source code appropriately for the respective platform. For the code presented here this has already been done, when Perl and Gtk-Perl were installed, so this step is dropped and the data volumes remain more manageable. Also the syntax leans heavily on the normal Gtk+ and GNOME under C, so the methods on objects all have matching designations. Support for Perl by Gtk-Perl is not however restricted to simple GUI functions, but also extends to image processing by *Imlib* and *GdkPixBuf*, to the processing of Glade data and also to more exotic widgets like *GtkHTML* and *GtkGLArea*. Coding examples are on the Web – for real this time. Happy Gnoming! ■

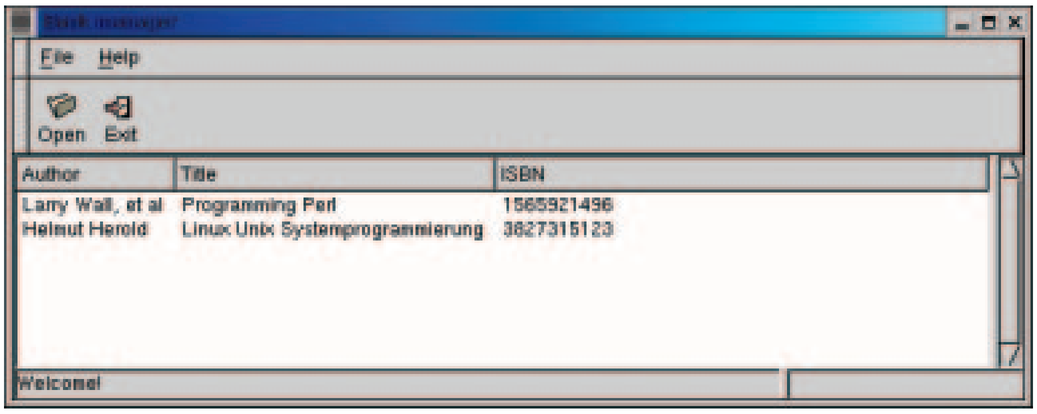
Info

Gnome Website: http://www.gnome.org/
CPAN: http://www.cpan.org/
Gtk-Perl Homepage: http://projects.prosa.it/gtkperl/
Thorsten Fischer: GUI-programierung mit Gtk+, SuSE-Press 2000
Code examples from the article: http://www.derfrosch.de/weichewaren/linux-magazin.html

■

The author

Thorsten Fischer is a student of computer science and Media consultancy at the Technical University of Berlin, his book "GUI-programierung mit GTK+" was published in October just in time for the book fair by SuSE-Press. He also works as a developer for Mapmedia in Berlin.



Author	Title	ISBN
Larry Wall, et al	Programming Perl	1565921496
Helmut Herold	Linux Unix Systemprogrammierung	3827315123

Welcome!

Figure 2: Book manager