EuroForth 2017
In Cahoots
Forth, GTK and Glade working secretly together

N.J. Nelson B.Sc. C. Eng. M.I.E.T.
R.J. Merrett B.Eng.
Micross Automation Systems
Unit 6, Ashburton Industrial Estate
Ross-on-Wye, Herefordshire
HR9 7BW UK
Tel. +44 1989 768080
Emails: njn@micross.co.uk rjm@micross.co.uk

Abstract
Forth is a very good language for working with other tools and libraries. In this paper we will introduce some techniques to make GTK and Glade work with Forth as seamlessly as possible.

1. Introduction
Cahoots in this instance does not refer to that well-known town in New York state on the banks of the Hudson River.[1] Its alternative meaning is when two or more parties conspire to act together secretly. The parties in this case are:

Forth

Our favourite language for conciseness, readability, and in this case ease of interoperability.

GTK

This is one of several open source toolkits for graphical programming.
It was chosen because it is being very actively developed, and has a straightforward interface method.

Glade

This is a graphical design tool for GTK. It produces XML code that can be loaded by GTK as required.

2. Compilers, versions and targets
We have been working with the MPE VFX Forth compiler, using GTK+ version 3, for the Ubuntu Linux operating system on both single-board computers with ARM processors, and industrial PCs with x86 CPUs.

1 Roger S. Brody RDP, Chairman of the Smithsonian Museum Philatelic Research Committee.

3. Library bindings

The MPE compiler came with a basic set of bindings to the GTK+ V2 libraries. We adapted these to GTK+ V3, added many new bindings and enumerations as needed, and removed features that we concluded were dead ends. Since GTK+ is written in C, the bindings are very straightforward e.g.

```
extern: void  "c" gtk_button_set_image( int * button, int * image );
```

4. Maintaining interactivity

A major difference between MPE VFX for Linux and MPE VFX for Windows, is that the Linux version runs straight from a standard Linux terminal. This means that interactivity is lost as soon as the GTK message pump starts. Since we regard interactivity as an essential debugger tool, it was necessary to restore it somehow.

Because there are always small differences needed in behaviour between programs when run in debug and when run normally (e.g. so that logon is not necessary every time you run in debug), we always create two different build files which set or clear a debugging flag e.g.

Debug

```
TRUE VALUE DEBUGGING          \ Set debugging mode
include PackingLabel.bld      \ Main build file
.BadExterns                   \ Report any library failures
PACKINGLABELMODULE            \ Run in debug
```

Compile

```
FALSE VALUE DEBUGGING         \ Clear debugging mode
include PackingLabel.bld      \ Main build file
PACKINGLABELMODULE            \ Run, to get it all set up
save PackingLabel             \ Save ELF file
```

This debugging flag can then be used to start the GTK message pump in a separate thread, when in debug mode.

```
TASK MAINTASK \ For GTK in debug
: MAINACTION ( --- ) \ GTK action, when in debug
  gtk_main              \ Start the message pump
;
```

```
...
  INIT-MULTI                              \ Initialise the multitasker
  MULTI                                   \ Start the multitasker
  DEBUGGING IF                            \ Running in debug
    ['] MAINACTION MAINTASK INITIATE      \ Start main in separate thread
  ELSE
    gtk_main                              \ Start the message pump
  THEN
...
```

Forth commands can then continue to be run from the Linux terminal, when in debug mode.

5. Structuring the Glade files

The VFX Forth comes with a nice wrapper which both loads the Glade XML file, and resolves the signals, in one operation. However, this is restricted to a single Glade file, and in a real application a single Glade file soon becomes too big to handle. We started to split the files by function, which also makes for re-usability. However, a single builder object is used for all the Glade files, so that all windows, dialogs and other features can be handled together. We also separated the file load from the signal resolution, because of the next feature.

```
: LOADGLADE { | be[ cell ] -- } \ Loads the glade files
  gtk_builder_new -> PBUILDER                              \ Create builder
  PBUILDER IF
    be[ OFF
    PBUILDER Z" SW1015.glade" be[ gtk_builder_add_from_file      \ Main glade
    PBUILDER Z" Logon.glade"  be[ gtk_builder_add_from_file AND  \ Logon glade
\   PBUILDER Z" next file .." be[ gtk_builder_add_from_file AND  \ More ...
    0= IF
      be[ @ 2 cells + @ .z$                                    \ Error string
      be[ @ g_error_free
      PBUILDER g_object_unref
    THEN
  THEN
;
```

6. Handling the handles

In order to do anything with a GTK+ widget, you need to know its magic number - the equivalent of a "handle" in Windows. When designing in Glade, you specify a name, then at run time you can ask the "builder" into which you loaded the Glade file, for the number of an object, from its name. You can then store it in a value (typically of the same name).

```
: GETHANDLE ( z$---h ) \ Get handle of builder element from name
  PBUILDER SWAP gtk_builder_get_object
;
```

```
Z" Mybutton" GETHANDLE -> MYBUTTON
```

That was OK for simple applications, but then we soon realised that we were typing the name of every widget three times before we even used it - once in the Glade design, once to declare the value, and once to get the magic number.

In any other language other than Forth, you are stuck with that.

But as so often happens, the unique ability of Forth to do things during compilation time as well as during run time, comes to the rescue. We soon discovered that it's possible to get the builder to create a list of all objects, which can then be scanned for names.

```
: MAKEGLADENAMES { | pslist pobject -- } \ Create values for every object
  PBUILDER gtk_builder_get_objects -> pslist   \ Make list of objects
  pslist g_slist_length 0 ?DO                 \ For all objects
    pslist I g_slist_nth_data -> pobject      \ Get data
    pobject gtk_buildable_get_name            \ Get name
    pobject ZVALUE                            \ Create value for each name
  LOOP
  pslist g_slist_free                         \ Free list
;
```

This uses a very cunning feature - the ability to create Forth values automatically.

```
: ZVALUE ( zname, ival --- ) \ Creates a new value with name and initialisation
  SWAP ZCOUNT ($CREATE)
    , ['] valComp, set-compiler
  interp>
    valInterp
;
```

Note: you need an up-to-date version of VFX to make this work.

All that is necessary during a debug, is to call both LOADGLADE and MAKEGLADENAMES during the compile, and all the values are ready for you to use. However, when you then run an executable, you've got the value names, but not their magic numbers. It's necessary to distinguish between debug and normal run mode again, to load the numbers when necessary.

```
: SETGLADEVALS { | pslist pobject -- } \ Set values for glade objects
  PBUILDER gtk_builder_get_objects -> pslist   \ Make list of objects
  pslist g_slist_length 0 ?DO                  \ For all objects
    pslist I g_slist_nth_data -> pobject       \ Get data
    pobject gtk_buildable_get_name             \ Get name
    zcount search-context IF                   \ Name is in dictionary
      >body pobject SWAP !                     \ Set value
    THEN
  LOOP
  pslist g_slist_free                          \ Free list
;
```

Notice that it's rather important to make sure the Glade widget names are Forth-unique, otherwise strange things happen.

Now in our initialisation, we simply include

```
...
  PBUILDER 0= IF              \ Glade not loaded
    do_gtk_init               \ Initialise GTK
    LOADGLADE                 \ Load glade files
    SETGLADEVALS              \ Set values for glade objects
  THEN
  RESOLVEGLADE                \ Resolve Glade signals
...
```

7. To do - automatic resizing

Most of the applications that we have written recently have been for touchscreens, in "kiosk" mode i.e. the operator has no access to the underlying operating system. This is far easier to achieve in Linux than it is in Windows, where it has become more and more difficult to eliminate the infuriating little things that Windows "pops up" without being asked.

Of course, any kiosk applications must run full screen. But screen resolution may vary. In Windows, the size and position of each element is under the exact control of the programmer. We used to design each display based on the minimum plausible resolution (say, 800 x 600 pixels) then use a Forth word that ran through all possible windows, and resized and repositioned them according to the actual screen resolution. The fonts were also resized to match the vertical resolution.

```
: CTRL2RES { ahctrl -- } \ Set size and position of control
  ahctrl HIROANIM @ =                     \ Animation control
  ahctrl HIRONSETUP @ = OR IF             \ or, superimposed button
    ahctrl                                \ Move only, do not size
    ahctrl WINDOW-X ahctrl WINDOW-Y 0 0
    RESVAR-XYWH 2DROP WINDOW-AMOVE
  ELSE                                    \ All other controls
    ahctrl 0                              \ Resize and move
    ahctrl WINDOW-X ahctrl WINDOW-Y
    ahctrl WINDOW-WIDTH ahctrl WINDOW-HEIGHT
    RESVAR-XYWH
    SWP_NOZORDER SWP_NOSENDCHANGING OR
    WINSETWINDOWPOS DROP
    ahctrl RESVAR-FONT                    \ New font
  THEN
;
```

```
: WIN2RES ( Whndl --- ) \ Set size and position of window and all controls
  DUP 0 0 WINDOW-AMOVE
  DUP CURRHORZRES @ CURRVERTRES @ WINDOW-ASIZE
  DUP RESVAR-FONT
  WINGETTOPWINDOW ?DUP IF
    BEGIN
      DUP CTRL2RES
      GW_HWNDNEXT WINGETNEXTWINDOW ?DUP
      0=
    UNTIL
  THEN
;
```

Unfortunately, this is not so easy in GTK. There is a heavy emphasis on automatic sizing of widgets. Before rendering, each container widget asks all the contained elements right down the chain, for the size they'd like to be. This can be a minimum size that has been set in Glade, but it is usually not possible to set a maximum size. So if you have a label widget, and increase the length of its string or the size of its font, and it will automatically resize itself, which in turn will resize its container, and so on up the chain. If the topmost window is now too big for the screen resolution, it will create scrollbars for itself, and worse still, reveal the Ubuntu toolbar.

We have still not fully resolved this problem, and for the time being there is the very irritating and time-consuming process of making a different set of Glade files for each screen resolution.

We're sure we cannot be the only people with this issue, and suggestions are very welcome.

8. Conclusion

Only in Forth, can one successively improve the compilation process so that each application becomes more compact and easier to write.

NJN
RJM
30/8/17