IUP, Portable User Interface

Antonio Scuri

The author is a senior developer at the Computer Graphics Technology Group (Tecgraf) of the Pontifical Catholic University of Rio de Janeiro (PUC-Rio). He is responsible for the Portable User Interface toolkit (IUP), the Canvas Draw graphics library (CD), the Imaging toolkit (IM), and other tools for desktop application development.

Contact with the author: scuri@*tecgraf.puc-rio.br*

Abstract

IUP is a free portable toolkit for building graphical user interfaces. It uses a very flexible license and can be used for public and commercial applications. The library and its API are implemented in C, but it has a binding to the Lua language.

IUP purpose is to allow the user interface of a program to be executed in different systems without any modification. The toolkit is available for Motif and Windows native interfaces. Pre-compiled binaries are available for several compilers and systems: GCC and CC, in UNIX environments (SunOS, IRIX, AIX, FreeBSD and Linux); Visual C++, Borland C++, Watcom C++ and GCC (Cygwin and MingW), in Windows environment.

Its main advantages are high performance, due to the fact that it uses native interface elements, and a fast learning by the user, due to the simplicity of its API. It also offers the possibility to customize applications, due to both the dialog specification language (LED) and the Lua binding (IupLua) use of a simple text file. IUP uses an abstract layout model based on the boxes-and-glue paradigm from the TEX text editor. This model, combined with LED or IupLua makes the dialog creation task more flexible and independent from the graphics system's resolution.

This article will present a small tutorial on how to create an application interface using IUP.

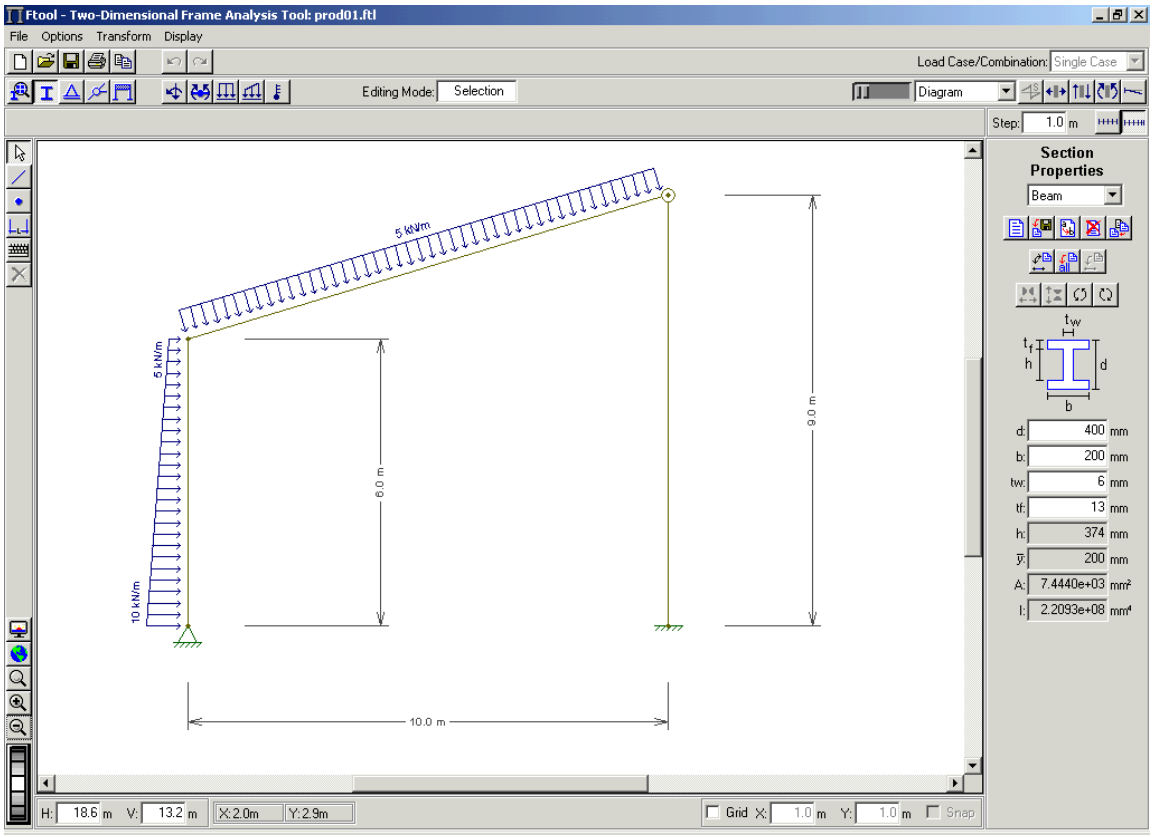

Figure 1. Two-dimensional Frame Analysis Tool

Getting Started

IUP has four important concepts that are implemented in a very different way from other toolkits.

First is the control creation time line. When a control is created it is not immediately mapped to the native system. So some attributes will not work until the control is mapped. The mapping is done when the dialog is shown or manually calling IupMap for the dialog. You can not map a control without inserting it into a dialog.

Second is the attribute system. IUP has only a few functions because it uses string attributes to access the properties of all controls. So get used to IupSetAttribute and IupGetAttribute, because you are going to use them a lot.

Third is the abstract layout positioning. IUP controls are never positioned in a specific (x,y) coordinate inside the dialog. The positioning is always calculated dynamically from the abstract layout hierarchy. So get used to the IupFill, IupHbox and IupVbox controls that allows you to position the controls in the dialog.

Fourth is the callback system. Because of the LED resource files IUP has an indirect form to associate a callback to a control. You must perform two steps: associate a C function with a name

using IupSetFunction, and associate the callback attribute with this name using IupSetAttribute.

LED is the original IUP resource file which has been deprecated in favor of IupLua files. But keep in mind that you can use IUP using only the C API.

Initialization

Before calling any IUP function, IupOpen must be called to initialize the toolkit. At the end IupClose should be called so that the toolkit can free internal memory and close the interface system. Therefore, usually an application employing IUP will have a code in the main function similar to the one in Listing 1. To use IUP in Lua, additional initialization is necessary for Lua and for the IupLua binding.

Listing 1. Initialization

```
#include <iup.h>

void main(void)
{
   if (IupOpen() == IUP_ERROR)
   {
      fprintf(stderr, "Error Opening IUP.");
      return;
   }

   appMainDialog();

   IupClose();
}
```

Hello World!

The traditional “Hello World” program is quite simple, just replace appMainDialog() in Listing 1 by IupMessage(“IUP”, “ Hello World!”). But it will be more interesting if you build your own dialog to display the message.

In IUP you will first create the elements that compose the dialog and then you create and show the dialog. For this dialog you will need a label for the message text and a button to close the dialog.

Listing 2 shows the implementation of the appMainDialog() function and Figure 2 its result (the font size was increased to improve print resolution). Listing 3 shows the equivalent code implemented in Lua.

The IUP dialog accepts only one child so the two controls must be inserted in a container, in this case a vertical box. The default alignment for a vertical box is at left, so it is changed to center.

Also margins and gap were set for the box.

To show the dialog IupShow is used, it will map and show the dialog, and return to continue program execution. But to process the events of the interface system IupMainLoop should be called at least once, there the execution will stop. To end the event loop the callback cbOK returns IUP_CLOSE. This callback is activated when the "OK" button is pressed.

If the application already have called IupMainLoop, or if you want to show a modal dialog, instead of using IupShow+IupMainLoop, you should use IupPopup that will show the dialog and stops the execution until some callback returns IUP_CLOSE. In this case only the IupPopup will be terminated.

Ate the end the dialog should be destroyed to avoid memory leaks. Notice that only the dialog is destroyed, the other child controls are automatically destroyed.

Listing 2. Hello World!

```
int cbOK(Ihandle* self)
{
   return IUP_CLOSE;
}

void appMainDialog(void)
{
   Ihandle *dlg, *vbx, *lbl, *btn;

   lbl = IupLabel("Hello World!");
   /* sets title and callback name */
   btn = IupButton("OK", "cbOK");

   /* associates callback name to C function */
   IupSetFunction("cbOK", cbOK);

   vbx = IupVbox(lbl, btn, NULL);
   IupSetAttribute(vbx, "MARGIN", "15x15");
   IupSetAttribute(vbx, "GAP", "10");
   IupSetAttribute(vbx, "ALIGNMENT", "ACENTER");

   dlg = IupDialog(vbx);
   IupSetAttribute(dlg, "TITLE", "IUP");

   IupShow(dlg);

   /* The execution will stop here
      to process events. */
   IupMainLoop();

   /* Only the dialog is destroyed. */
   IupDestroy(dlg);
}
```

Listing 3. Hello World! in Lua

```
function cbOK()
   return iup.CLOSE
end

dlg = iup.dialog
{
   iup.vbox
   {
      iup.label{title="Hello World!"};
      iup.button{title="OK", action=cbOK},
      gap="10",
      alignment="ACENTER",
      margin="15x15"
   };
   title="IUP"
}

dlg:show()

iup.MainLoop()

dlg:destroy()
```

Figure 2. Hello World!

Beyond Hello World

Now that you know how attributes and callbacks work, lets take a look at a more rich example. This “typical” application will include a menu, a toolbar, some controls and a canvas. Listing 4 shows the implementation and Figure 3 its result.

The example demonstrates several aspects of IUP attributes and callbacks with a more complex layout than the Hello World example. The layout uses a vertical box to include two horizontal boxes: one for the “toolbar” of controls, and one for the radio frame and the canvas. The example also introduces two additional resources: images and menus; and shows how to manage

one control from inside the callback of another control.

Listing 2. Beyond Hello World

```
/* this is the raw image data.
   Images can be created in C, LED, Lua or
   loaded from an image file using IupLoadImage. */
static char img_bits[] =
{
 0,0,0,0,0,0,0,2,1,1,1,2,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,2,1,1,1,2,0,0,0,0,0,0,0
,0,0,0,0,0,0,2,1,1,1,1,1,2,0,0,0,0,0,0
,0,0,0,0,0,0,2,1,1,1,1,1,2,0,0,0,0,0,0
,0,0,0,0,0,2,1,1,1,1,1,1,1,2,0,0,0,0,0
,0,0,0,0,2,2,1,1,1,1,1,1,1,2,2,0,0,0,0
,0,0,2,2,1,1,1,1,1,1,1,1,1,1,2,2,0,0,0
,2,2,1,1,1,1,1,1,2,2,2,1,1,1,1,1,1,2,2
,1,1,1,1,1,1,1,2,2,2,2,2,1,1,1,1,1,1,1
,1,1,1,1,1,1,1,2,2,2,2,2,1,1,1,1,1,1,1
,1,1,1,1,1,1,1,2,2,2,2,2,1,1,1,1,1,1,1
,2,2,1,1,1,1,1,1,2,2,2,1,1,1,1,1,2,2,2
,0,0,2,2,1,1,1,1,1,1,1,1,1,1,2,2,0,0,0
,0,0,0,0,2,2,1,1,1,1,1,1,1,2,2,0,0,0,0
,0,0,0,0,0,2,1,1,1,1,1,1,1,2,0,0,0,0,0
,0,0,0,0,0,0,2,1,1,1,1,1,2,0,0,0,0,0,0
,0,0,0,0,0,0,2,1,1,1,1,1,2,0,0,0,0,0,0
,0,0,0,0,0,0,0,2,1,1,1,2,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,2,1,1,1,2,0,0,0,0,0,0,0
};

int cbExit(Ihandle* self)
{
   return IUP_CLOSE;
}

int cbAbout(Ihandle* self)
{
   IupMessage("About", "IUP Beyond Hello World 1.0");
   return IUP_DEFAULT;
}

int cbColorList(Ihandle* self, char *t, int i, int v)
{
   /* when an item is selected */
   if (v == 1)
   {
      /* attribute inheritance will find the "appCanvas" in the
dialog */
      Ihandle* cnv = (Ihandle*)IupGetAttribute(self, "appCanvas");
      switch(i)
      {
      case 1:
         IupSetAttribute(cnv, "BGCOLOR", "255 0 0");
         break;
      case 2:
```

```
         IupSetAttribute(cnv, "BGCOLOR", "0 255 0");
         break;
      case 3:
         IupSetAttribute(cnv, "BGCOLOR", "0 0 255");
         break;
      case 4:
         IupSetAttribute(cnv, "BGCOLOR", "255 255 255");
         break;
      }
   }
   return IUP_DEFAULT;
}

int cbReset(Ihandle* self)
{
   Ihandle *lst, *cnv, *lbl;

   /* attribute inheritance will find the "appCanvas" in the dialog
*/
   cnv = (Ihandle*)IupGetAttribute(self, "appCanvas");
   IupSetAttribute(cnv, "BGCOLOR", "255 255 255");

   lst = (Ihandle*)IupGetAttribute(self, "appList");
   IupSetAttribute(lst, "VALUE", "4");

   lbl = (Ihandle*)IupGetAttribute(self, "appLabel");
   IupSetAttribute(lbl, "TITLE", "");

   return IUP_DEFAULT;
}

int cbButton(Ihandle* self,int but,int press,int x,int y,char*
status)
{
   Ihandle* lbl = (Ihandle*)IupGetAttribute(self, "appLabel");
   IupSetfAttribute(lbl, "TITLE", "button_cb(%d,%s,%d,%d)", but,
press?"press":"release", x, y);
   return IUP_DEFAULT;
}

int cbKeyPress(Ihandle *self, int c, int press)
{
   if (press)
   {
     Ihandle* lbl = (Ihandle*)IupGetAttribute(self, "appLabel");
     IupSetfAttribute(lbl, "TITLE", "keypress_cb(%c)", (char)c);
   }
   return IUP_DEFAULT;
}

int cbMotion(Ihandle *self, int x, int y, char *r)
{
   Ihandle* lbl = (Ihandle*)IupGetAttribute(self, "appLabel");
   IupSetfAttribute(lbl, "TITLE", "motion_cb(%d,%d)", x, y);
   return IUP_DEFAULT;
}

int cbShow(Ihandle* self, int v)
```

```
{
   if (v)
   {
      Ihandle* lbl = (Ihandle*)IupGetAttribute(self, "appLabel");
      IupSetAttribute(lbl, "TITLE", "");

      /* removing the callback association, the callback is not
called. */
      IupSetFunction("cbButton", NULL);
      IupSetFunction("cbMotion", NULL);
      IupSetFunction("cbKeyPress", NULL);

      switch (IupGetInt(self, "appToggleId"))
      {
      case 1:
        IupSetFunction("cbMotion", cbMotion);
        break;
      case 2:
        IupSetFunction("cbButton", cbButton);
        break;
      case 3:
        IupSetFunction("cbKeyPress", cbKeyPress);
        break;
      }
   }
   return IUP_DEFAULT;
}

void appCreateMainDialog(void)
{
   Ihandle *dlg, *mainbx, *cnv, *frm, *lbl,
           *toolbx, *mnu, *img, *lst;

   img = IupImage(19,19, img_bits);
   /* The BGCOLOR value indicates that
      the background color of the button
      should be used as the color value. */
   IupSetAttribute (img, "0", "BGCOLOR");
   IupSetAttribute (img, "1", "0 0 0");
   IupSetAttribute (img, "2", "255 64 64");
   /* the image will be associated with the button using a name */
   IupSetHandle ("imgReset", img);

   mnu = IupMenu(
      IupSubmenu("File",
         IupMenu(
            IupItem("Exit", "cbExit"),
            NULL)),
      IupSubmenu("Help",
         IupMenu(
            IupItem("About...", "cbAbout"),
            NULL)),
      NULL);
   /* the menu will be associated with the dialog using a name */
   IupSetHandle("mnuMain",mnu);

   IupSetFunction("cbExit", cbExit);
   IupSetFunction("cbAbout", cbAbout);
```

```
   /* The IupSetAttributes functions helps to build the layout
      because it returns the control handle. */
   toolbx = IupHbox(
      IupSetAttributes(IupButton("", "cbReset"), "IMAGE=imgReset,
EXPAND=NO"),
      IupSetAttributes(IupButton("About...", "cbAbout"),
"EXPAND=NO"),
      IupSetAttributes(IupLabel(""), "SEPARATOR=VERTICAL,
EXPAND=VERTICAL"),
      IupSetAttributes(lst = IupList("cbColorList"), "DROPDOWN=YES,
1=Red, 2=Green, "
                                                  "3=Blue, 4=White,
VALUE=4, EXPAND=NO"),
      IupSetAttributes(lbl = IupLabel(""), "EXPAND=YES"),
      NULL);
   IupSetAttribute(toolbx, "ALIGNMENT", "ACENTER");
   /* because the EXPAND attribute is set here
      it has to be removed from all the child controls
      that are not to expand. */
   IupSetAttribute(toolbx, "EXPAND", "HORIZONTAL");
   IupSetAttribute(toolbx, "GAP", "10");

   IupSetFunction("cbReset", cbReset);
   IupSetFunction("cbColorList", cbColorList);

   frm = IupFrame(
      IupRadio(
         IupVbox(
            IupSetAttributes(IupToggle("Show Mouse Move", "cbShow"),
"appToggleId=1"),
            IupSetAttributes(IupToggle("Show Mouse Button",
"cbShow"), "appToggleId=2"),
            IupSetAttributes(IupToggle("Show Keys", "cbShow"),
"appToggleId=3"),
            NULL)));
   IupSetAttribute(frm, "TITLE", "Options");

   /* use only one callback for the 3 toggles. */
   IupSetFunction("cbShow", cbShow);

   cnv = IupCanvas("");
   /* this will be the minimum size of the canvas. */
   IupSetAttribute(cnv, "RASTERSIZE", "280x200");
   IupSetAttribute(cnv, "BUTTON_CB", "cbButton");
   IupSetAttribute(cnv, "MOTION_CB", "cbMotion");
   IupSetAttribute(cnv, "KEYPRESS_CB", "cbKeyPress");

   IupSetFunction("cbMotion", cbMotion);

   mainbx = IupVbox(
      toolbx,
      IupHbox(
         cnv,
         frm,
         NULL),
      NULL);
   IupSetAttribute(mainbx, "MARGIN", "5x5");
```

```
    IupSetAttribute(mainbx, "GAP", "5");

    dlg = IupDialog(mainbx);
    IupSetAttribute(dlg, "TITLE", "IUP");
    IupSetAttribute(dlg, "MENU", "mnuMain");
    /* store some controls as an attribute to retrieve them in the
callbacks */
    IupSetAttribute(dlg, "appCanvas", (char*)cnv);
    IupSetAttribute(dlg, "appList", (char*)lst);
    IupSetAttribute(dlg, "appLabel", (char*)lbl);

    IupShow(dlg);

    IupMainLoop();

    IupDestroy(dlg);
    /* the image can be reused many times,
       but it must be destroyed once. */
    IupDestroy(img);
}
```

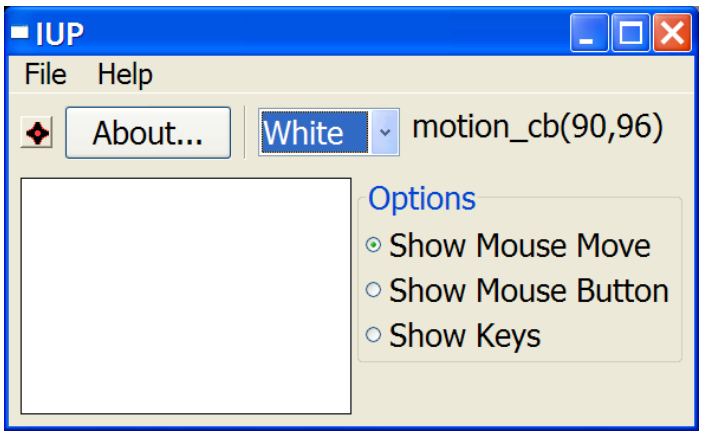

Figure 3. Beyond Hello World

Additional Controls

All the standard controls use native controls of the window system, in Microsoft Windows or in Motif. But IUP has also some additional controls that are build on top of IupCanvas, therefore they are drawn and managed by IUP, not by the native system. They are drawn using the Canvas Draw (CD) graphics library. The two most important controls are the IupMatrix and the IupTree. IupMatrix is a very powerful matrix of alphanumeric fields and IupTree is a tree containing nodes of branches or leaves composing a visual hierarchy. To use the additional controls you must initialize the additional library calling IupControlsOpen just after IupOpen.

What's Missing?

Some years ago there was one major user interface toolkit in UNIX systems, the Open Software Foundation Motif. Nowadays, this is still true for commercial UNIX systems, but for the free software community several toolkits are de-facto standards for user interface construction. The two major toolkits are the GTK+ (basis for the GNOME desktop) and the Qt (basis for the KDE desktop). Since GTK+ is in C, it is perfect to be used for a IUP driver, offering a more “native” look and feel than Motif in Linux. So this is the next big issue for IUP future version.

Some new controls are also planned, like: image lists, HTML viewer, and a 2D graph plot.

On the Web

- IUP - A Portable Graphical Interface Toolkit
  http://www.tecgraf.puc-rio.br/iup/
- CD - Canvas Draw, a 2D graphics library
  http://www.tecgraf.puc-rio.br/cd/
- The Programming Language Lua
  http://www.lua.org/

Final Remarks

If you consider Portability, Free License and Open Source, Small and Simple API, and Native Look and Feel, then you have the characteristics that make IUP a unique User Interface toolkit. The only toolkit with similar characteristics is wxWidgets. Its API is in C++, it has more controls than IUP, but it can be considered a more complex toolkit.

IUP is in C, is small and powerful. We have a small but very active team and we have many Tecgraf and foreign applications that today use IUP, collaborating for its evolution.

This work was developed at Tecgraf by means of the partnership with PETROBRAS/CENPES.

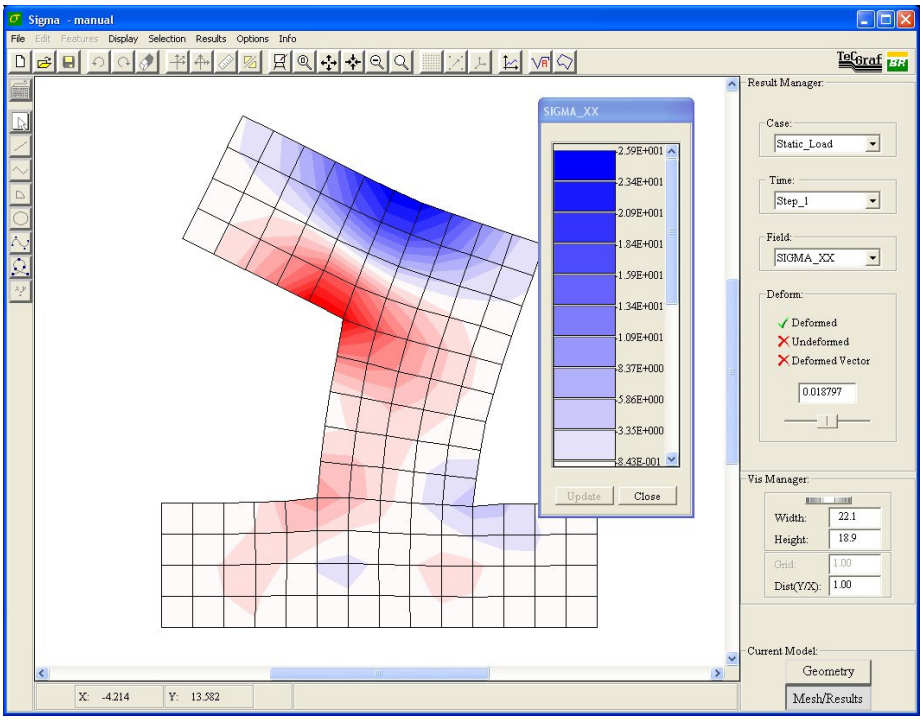

Figure 4. Integrated Geotechnics System for Multiple Analysis