Stanford Exploration Project, Report 67, January 13, 1998, pages 1–158

Xvpanel: interactive programs in one line or less

Steve Cole[1]

keywords: *algorithm, interactive*

INTRODUCTION

In SEP–65, I described an interactive data processing environment that I developed called IPE. Briefly, IPE allows you to use existing batch programs in a pseudo-interactive manner, complete with buttons and sliders that can be used to change parameter values. No programming is required; all that is needed is an ASCII description of the parameters for each program.

While IPE is fine for many purposes, in other cases it is overkill. This became clear during the development of interactive documents at SEP (Karrenbach and Nichols, 1990). If we want to let users modify one or two parameters in the processing sequence used to generate a particular figure, IPE, which shows the user *all* the parameters of *all* the programs in the processing sequence, confuses the issue unnecessarily.

In this paper, I discuss an X-Windows utility called *xvpanel* that I have written to address this need. Xvpanel allows the user to create a panel (similar in appearance to IPE but more modest) that runs a job interactively. Xvpanel's advantage is not only that the result is more compact, but also that *all* of the information needed to construct the interactive panel is specified on the command line with a few simple arguments. Thus it is easier to use than IPE. Also, since isn't built upon a pipelined approach to processing seismic data, it may have a wider range of applicability.

EXAMPLES

Here is an example to illustrate the use of xvpanel. If I type on the command line (I have separated it here onto several lines for clarity):

```
xvpanel
button dir pwd
choice "disk space" 2 /scr "df /scr" /scr2 "df /scr2"
```

[1]**email:** not available

The panel shown in Figure 1 is displayed on my screen. Pressing the button marked **dir** causes the Unix command *pwd* (print working directory) to be executed to tell me what directory I'm working in. Clicking on either **/scr** or **/scr2** invokes the *df* command (disk free) to find out how much space is available on either disk units.

The call to xvpanel has, in effect, created a small interactive program that performs either of these two functions. The main selling point of xvpanel is that *no* programming had to be done to produce this panel. All the work was done on the command line. No knowledge of X Windows, XView, or any programming language was required.

Below is a listing of a scaled-down version of xvpanel.c. This 112 line version handles only the two types of objects shown in Figure 1, buttons and multiple-choice items. The full program, including comments, is about 450 lines long, which is a bit too long to include in this paper. Following the listing are some comments that explain some of the workings of the program and of XView programs in general.

```
#include <stdio.h>
#include <xview/xview.h>
#include <xview/panel.h>

#define BUTTON          "button"
#define CHOICE          "choice"

Frame           base_frame;
Panel           control_panel;

main(argc,argv)

int argc;
char **argv;
{
  Panel_item item;
  int iargc=1;
  char *label;
  void quit_notify();
  void button_notify();
  void choice_notify();
  int i,numargs,val,valmin,valmax,length,width;
  int irow=0,icol=1;

  /* initialize XView */
  xv_init(XV_INIT_ARGC_PTR_ARGV,&argc,argv,NULL);

  /* create frame */
  base_frame = xv_create(NULL, FRAME,
               FRAME_LABEL,           "xvpanel",
               FRAME_NO_CONFIRM,      TRUE,
               NULL);

  /* create panel */
  control_panel = xv_create(base_frame,PANEL,NULL);
```

```
  /*
   * loop over command line arguments. search for a valid parameter type.
   * valid types are: button, choice.
   * then read other arguments to create a panel item.
   */
  while (iargc < argc)
  {
        /* button */
        if (!strcmp(argv[iargc],BUTTON))
        {
                item = (Panel_item) xv_create(control_panel,PANEL_BUTTON,
                        PANEL_LABEL_STRING,     argv[iargc+1],
                        PANEL_CLIENT_DATA,      argv[iargc+2],
                        PANEL_NOTIFY_PROC,      button_notify,
                        NULL);
                iargc+=3;
        }
        /* choice */
        else if (!strcmp(argv[iargc],CHOICE))
        {
                item = (Panel_item) xv_create(control_panel,PANEL_CHOICE,
                        PANEL_LABEL_STRING,     argv[iargc+1],
                        PANEL_NOTIFY_PROC,      choice_notify,
                        NULL);

                sscanf(argv[iargc+2],"%d",&numargs);
                for (i=0;i<numargs;i++)
                {
                        (void) xv_set(item,PANEL_CHOICE_STRING,i,argv[iargc+3+i*2],NULL);
                        (void) xv_set(item,XV_KEY_DATA,i,argv[iargc+4+i*2],NULL);
                }
                iargc+=(3+numargs*2);
        }
        else
        {
                fprintf(stderr,"xvpanel warning: ignoring unknown parameter type %s\n",argv[iargc]);
                iargc++;
        }
  }

  /* make panel just big enough to encompasss all items */
  window_fit(control_panel);
  window_fit(base_frame);

  /* pass control to notifier - wait for the user to do something */
  xv_main_loop(base_frame);
}

void
quit_notify(item,event)
Panel_item item;
Event *event;
{
```

```
        (void) xv_destroy_check(base_frame);
}

void
button_notify(item,event)
Panel_item item;
Event *event;
{
  char out[100];
  sprintf(out,"%-s",((char *) xv_get(item,PANEL_CLIENT_DATA)));
  fprintf(stdout,"%s\n",out);
}

void
choice_notify(item,value,event)
Panel_item item;
int value;
Event *event;
{
  char out[100];
  sprintf(out,"%-s",((char *) xv_get(item,XV_KEY_DATA,value)));
  fprintf(stdout,"%s\n",out);
}
```

Some comments about the code:

- All calls that begin with *xv_* are XView calls. The most important calls are *xv_create*, used to create an object, and *xv_set* and *xv_get*, used to set or query some attribute of an object. Most attributes can also be set at create time.
- An example of an attribute is PANEL_NOTIFY_PROC. This is the routine that gets called when a button or choice item is selected by the user. After all the items are created, xvpanel calls *xv_main_loop*. This passes control to the XView notifier, which displays the panel and calls the appropriate routine when one of the objects is selected.
- In IPE, I create a routine to be called for each object. Here I realized that I could avoid doing that by attaching any object-dependent data, such as the action associated with a particular button, to the object using the XView mechanisms PANEL_CLIENT_DATA and XV_KEY_DATA. These attributes allow arbitrary data to be attached to any XView object. PANEL_CLIENT_DATA can be used to attach one piece of information to an object. XV_KEY_DATA can be used to attach more than one piece of information. In the case of multiple-choice items, I needed to attach an action for every choice, so XV_KEY_DATA was required. For more information about these and other details of XView, see the XView Programming Manual (Heller, 1990).

Figure 2 is a more complicated example that illustrates some of xvpanel's features which are not included in the short version of the program given above. There are a few aspects of this example worth noting:

- This panel contains a **Doit** button. In cases such as Figure 1, where this button is absent, xvpanel responds with an action when any one of the objects is selected. Here, xvpanel responds only when the Doit button is pressed. At that time, it reads the values of all the parameters on the panel, and concatenates their actions together to form a single system call. The effect is much like IPE. A multi-program process can be run, with pipes passing the data from one step to the next. This is what is being done in Figure 2.

- Several types of objects not shown in the sample program are available — sliders, text fields, numeric-only text fields, and messages are used here. The newline item forces xvpanel to skip to the next line while building a panel. I have used it here to place the parameters for different programs on different lines. I have also used the message item, which has no associated action, to indicate the name of the program (Taplot, Ta2vplot, Xvpen) corresponding to each line.

- As with any X application, the title and other attributes of the xvpanel can be modified using standard command-line arguments.

The panel in Figure 2 was generated by typing:

```
xvpanel -label IPE-alike -system -doit -quit
message Input text "dataset name" "<" "/q2/wz/wz.25.H" 32
newline
message Taplot slider pclip " Taplot pclip=" 0 100 99
newline
message Ta2vplot choice color 3
grey "| Ta2vplot color=I "
clipped "| Ta2vplot color=IC "
flag "| Ta2vplot color=F "
newline
message Xvpen
numtext scale "| Xvpen scale=" 1 10 1
```

If the **Doit** button in Figure 2 were pressed, the resulting action taken by xvpanel would be:

```
< /q2/wz/wz.25.H Taplot pclip=99 | Ta2vplot color="I" | Xvpen scale=1
```

IPE UPDATE

IPE is in the process of being modified to take advantage of what I have learned while writing xvpanel. In the first version of IPE, which I described in SEP–65, an ASCII file for each program, describing its parameters, was read in by a program (Ipemake) that generated a C language subroutine to display a panel for that program. Thus each new program effectively had to be "compiled in" to IPE. Now, by attaching any program or parameter dependent information to objects using PANEL_CLIENT_DATA, I know how to dispense with this need for compiling in each program.

XVPEN UPDATE

I also used XView to write *Xvpen*, a graphics filter that displays SEP's vplot graphics on an X Windows screen. In IPE and in Figure 2, Xvpen is the natural final step in the processing sequence. In my paper, I pointed out that one drawback to the IPE/Xvpen combination was that it was slightly cumbersome. Each iteration of IPE generated a separate Xvpen window, and the user had to manually remove a window that was no longer needed. I expressed the hope that one day I would be able to simply re-use a window already on the screen. That has now been accomplished, and as a result both IPE and Xvpanel should be easier to use.

CONCLUSIONS

Xvpanel lets you bring an interactive panel up on the screen without having to program. All the customization is done right on the command line.

In one mode of operation, a system command can be built out of all the objects on the panel and their values. This is a scaled-down, "quick and dirty" alternative to my interactive processing environment, IPE. IPE has also benefited from my work on xvpanel and is becoming much easier to use.

In another mode of operation, a separate action is attached to each object on the panel. This may make xvpanel useful for purposes other than pipelined seismic processing.

A few problems remain with xvpanel. One is the length of the calling sequence needed to generate typical panels. While it can all be put on one line, as the title of my paper promised, it is quite cumbersome. Arguments can easily be forgotten or entered out of order. Perhaps I could provide some kind of interactive panel builder that prompts for parameters, or come up with a way to make things default better so that missing arguments are not fatal.

REFERENCES

Cole, S., 1990, An interactive processing environment: SEP–**65**, 305-314.

Heller, D., 1990, XView Programming Manual: O'Reilly & Associates, Inc.

Karrenbach, M., and Nichols, D., 1990, Progress towards the interactive book: SEP–**67**.