Chapter 1

Xtpanel - an interactive panel builder

This appendix describes a program called *xtpanel* that I wrote, along with Dave Nichols, to facilitate user interaction with existing non-interactive software. It is used extensively at SEP to allow readers to interact with documents such as this thesis, recreating the figures with the ability to try out new parameters or look at alternative pieces of data. I have included a description of xtpanel here because I feel it is a useful component in moving toward the goal of reproducible research that may be of interest to readers of this thesis.

The xtpanel program manages a set of interactive objects on the screen. These can either be specified on the command line or in a script file. The appendix contains the manual page for xtpanel, which describes the command line and script file syntax in more detail.

The following types of objects are supported:

- messages
- text fields
- dialog boxes
- sliders
- buttons
- menus
- lists
- toggle buttons
- variables (objects with no screen representation)

1.0.1 Object values

In xtpanel, each interactive object has a string value associated with it. Interacting with the screen representation of an object will change its string value (e.g., moving a slider sets the value to the value of the slider, typing in a dialog sets the value to what you type, choosing an item in a menu sets the value to the string associated with the menu selection.) The primary task of each object is to maintain its string value.

1.0.2 Object names

Every object may have a name. This name is used to refer to the value of the object in actions.

1.0.3 Object actions

Every object can also have a set of actions associated with it. These actions are performed whenever the object's string value is updated. If more than one action is specified for an object, they are performed in the order specified. The action can be one of five types:

QUIT Exit the xtpanel program.

PRINT Print a string to the standard output.

STRING Set the object's string value to a new value.

ASSIGN Set another object's string value to a new value.

SYSTEM Execute a command.

Enormous flexibility is gained by letting *any* object execute a command as the result of interaction with it. Xtpanel can be used as a way of interactively connecting existing Unix programs that do not have interactivity built into them. An impressive example of this is an 80 line xtpanel script that acts as an interactive front end to the Unix calculator program, bc. This 80 line program (see the Calculator section below) has similar functionality to the 2070 line C program `xcalc` distributed with X windows.

An action may contain two types of special characters.

1. The string value of any object can be used in an action string by referring to its name, preceded by a dollar sign. The following action string sets the value of its object to the current string value of another object whose name is `otherobj`:

   ```
   action="STRING $otherobj"
   ```

2. The string resulting from executing a command can be used in an action string by enclosing it in backquotes. For example, the following action string sets the value of an object to the current directory using the UNIX command `pwd`:

   ```
   action="PRINT The current directory is `pwd`"
   ```

The aim of this design is to allow the user to tie together existing Unix programs while leaving as much flexibility as possible.

1.1 Examples

The following pages contain example xtpanel script files and screen dumps of the corresponding interactive program. If you are reading this document on a CD all the figures are interactive, if you click on the button associated with a figure the program will be executed.

1.1.1 Simple buttons

The first script puts two buttons on the screen. Pressing one button quits xtpanel; hitting the second button prints a message. Figure 1.1 shows the panel produced.

```
button={ label="QUIT" action="QUIT" }
button={ label="hit me" action="PRINT AARGH!\n" }
```

Figure 1.1: A simple panel with two buttons. xtpanel-buttons [NR]

1.1.2 Simple menu

The second script implements a menu. The resulting panel is shown in figure 1.2.

```
button={ label=quit action=QUIT }
menubutton={ label=numbers action="PRINT choice is $val \n"
         item={ label=one value=1 }
         item={ label=two value=2 }
         item={ label=three value=3 }
       }
```

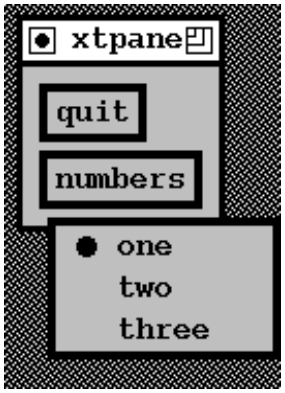

Figure 1.2: A simple panel with a menu and a quit button. xtpanel-menubutton [NR]

1.1.3 Simple interactive parameter selection

This exmaple shows how xtpanel can be used in conjunction with the cake rules for an interactive document (**?**) to make a dull figure come alive. The default cake rules for an interactive document check for the existance of a file called *name*.panel, whene *name* is the name of a figure. If this file exists the rule will run xtpanel when the button in the caption is pressed. Figure 1.3 shows a single synthetic event. When the user presses the button in the caption they are asked to choose an NMO velocity to flatten the event. Here is the xtpanel file that specifies the interactive program.

```
button={ label=QUIT action=QUIT }

slider={ label="Select a velocity for NMO"
     min=1 max=5 value=2 format="%.2f" width=300
    action="NMO <Dat/nmoin.H vel=$(val) | \
            Wiggle title=""velocity=$(val)"" pclip=100 | Tube numcol=16 &"
}

message={ value=" hit ""ok"" to run the program " }
```

When the user clicks on the OK button the data has NMO applied at the chosen velocity and the result is displayed on the screen. Fig 1.4 shows the interactive panel in use.

1.1.4 Simple seismic processing pipeline

The next example controls a simple processing job using SEP software. A dialog box is used to input a file name. This file is read into a bandpass filter program, then converted to one byte per sample (for display) by the program Byte, then plotted. Sliders are used to specify the cutoff frequencies for the filter, and for the percentile of the filtered data that is clipped to the maximum intensity in the float to byte conversion.

```
button={ name=QUIT action=QUIT }
button={ label=GO
  action="Bandpass flo=$flo fhi=$fhi phase=$phase < $file \
```

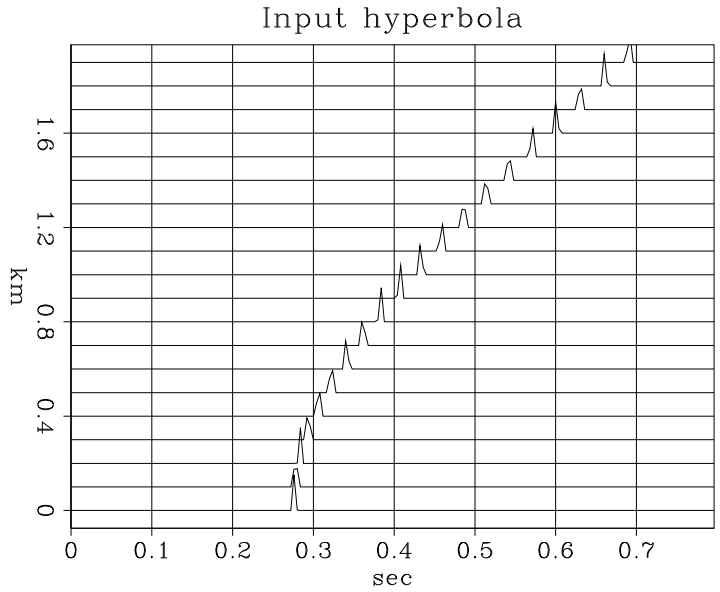

Figure 1.3: A synthetic dataset, click on the button to choose the correct NMO velocity. xtpanel-nmo [ER]

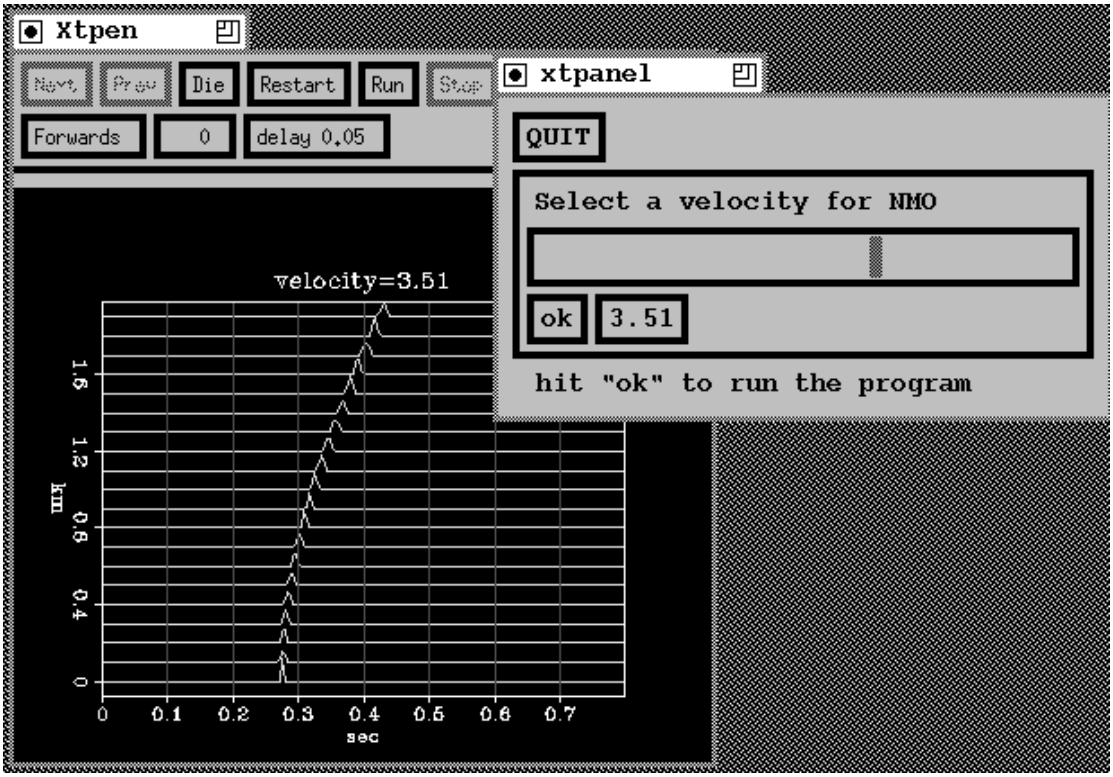

Figure 1.4: The simple velocity selection panel in use. xtpanel-nmores [NR]

```
            > /tmp/band.h out=/tmp/_band.h@; \
          Byte pclip=$pclip < /tmp/band.h | Ta2vplot | Xtpen; \
          /bin/rm /tmp/band.h &"
}
dialog={ name=file label="input file" value="Dat/wz.25.H" }
hbox={ name=noborder
  vbox={
    message={ value=Bandpass }
    slider={ name=flo label=flo min=0 max=125 value=0 format="%.2f" }
    slider={ name=fhi label=fhi min=0 max=125 value=125 format="%.2f" }
    choice={ name=phase label=phase value=0
      item={ label="zero" value="0" }
      item={ label="minimum" value="1" }
    }
  }
  vbox={
    message={ value=Byte }
    slider={ name=pclip label=pclip min=0 max=100 value=99 format="%.0f" }
  }
}
```

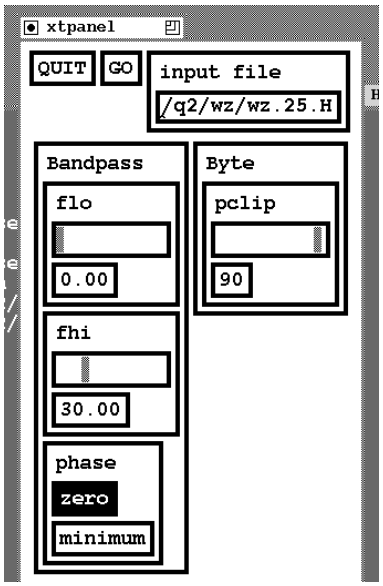

Figure 1.5: A panel to control the execution of two seplib programs. xtpanel-simplepipe [NR]

1.1.5 SEP data cube viewer

Here we use xtpanel not to process data, but to examine, sample by sample, the values contained in a dataset. From an input three-dimensional dataset, a single plane is selected (by a slider) and then the floating point values from that plane are displayed in a two-dimensional scrollable text field.

The first script gets the file name from the user:

```
button={ name=QUIT action=QUIT }
dialog={ name=file value="Dat/cube.H" }
button={ label=GO
```

```
  action="xtpanel -cpp -DFILE=$file -file viewer2.panel &"
}
```

and then calls a second script, which displays the data:

```
button={ name=QUIT action=QUIT }
dialog={ name=file value="Dat/cube.H" }
button={ label=GO
  action="xtpanel -cpp -DFILE=$file -file viewer2.panel &"
}
```

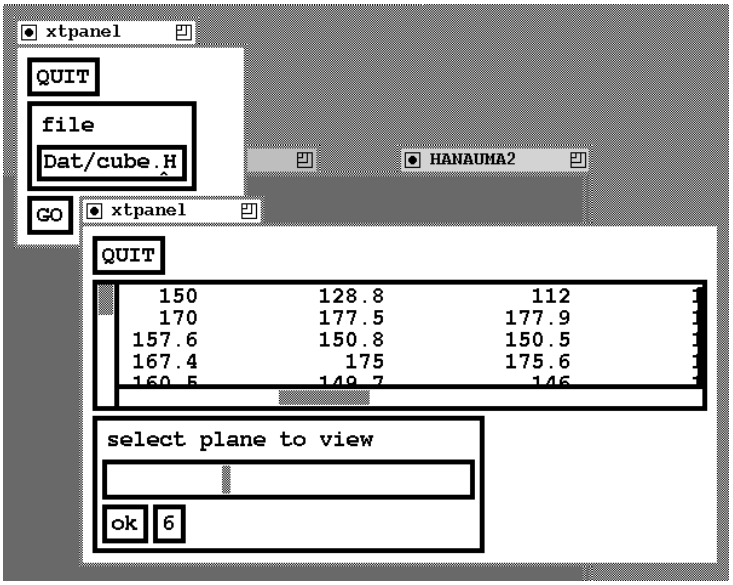

Figure 1.6: A viewer to examine SEP data cubes. xtpanel-viewer [NR]

Note that when the first script invokes the second, it uses the `-cpp` flag to pass the second script through the C preprocessor. This is one way to pass variables from one script to the next.

1.1.6 Directory lister

Now a more complicated script. This script can be used to navigate a directory hierarchy. It puts all the files in the current directory into a list. When you choose a file it does one of two things. If the file is a directory, it goes to that directory and reruns the program to produce a new listing panel. If the file is a regular file it performs the Unix command specified in the dialog. Note that the rules governing newlines within an action allow us to embed a complete multiline shell script within an action. The resulting panel is shown in figure 1.7.

```
button={ label=DONE action=QUIT }

hbox={
message={ value="Directory:" }
```

```
message={ value=‘pwd‘ }
}

dialog={ name=command label="Command for files" value="xterm -e vi" }

list={ name=name label="NAMES"
  action=" if test -d $val
           then
               cd $val
              xtpanel -file examples/script/lister &
           else
              $command $val &
           fi"
  itemlist={ list=‘echo -n .* * ‘ separator=" " }
}
```

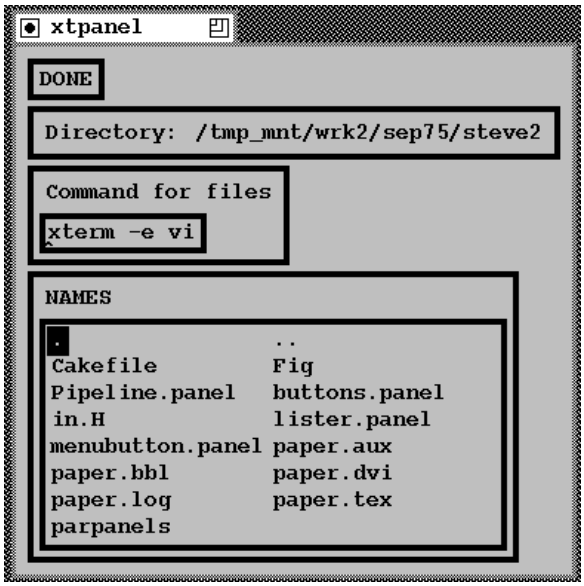

Figure 1.7: A panel to traverse a Unix filesystem. xtpanel-lister [NR]

1.1.7 Advanced seismic processing pipeline

Next is a multi script example. It is designed to allow the user to choose the parameters for a pipeline of three seplib processes. The main script is used to edit dialog fields containing parameters that specify the input file and the parameters for each of the three programs (Byte,Ta2vplot and Tube). Once the user is satisfied with the parameters he can press the button at the bottom of the panel to run the command. Figure 1.8 shows the main panel. Here is the corresponding script file. Notice that the layout for each program is the same, it is defined in a macro and the macro is used three times, once for each program.

```
button={ label=QUIT action=QUIT }

dialog={ name=input label="Input filename" value="Dat/in.H" }

! macro for a dialog for a program called "NAME", a popup panel and self doc
#define PROGRAM(NAME) \
```

```
hbox={ name=noborder \
  vbox={ name=noborder \
    button={ label="NAME panel" \
         action="ASSIGN NAME `xtpanel -file pars/NAME`"} \
    button={ label="NAME doc. " \
      action="xtpanel -cpp -file pars/doc -DPROG=NAME -font fixed &" \
    } \
  } \
  dialog={ name=NAME value="            " } \
}

PROGRAM(Byte)
PROGRAM(Ta2vplot)
PROGRAM(Tube)

! perform the action or popup an error dialog if the input doesn't exist
button={ label="PRESS FOR: Byte | Ta2vplot | Tube"
action="if test -f $input ; then
          <$input Byte $Byte | Ta2vplot $Ta2vplot | Tube $Tube &
        else
          xtpanel -message 'File $input does not exist' -quit
        fi "
}
```

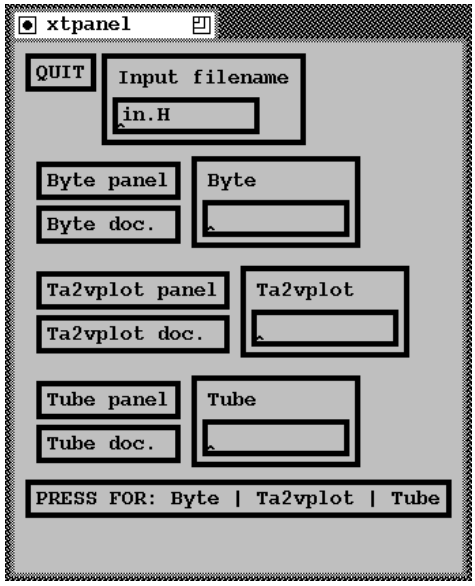

Figure 1.8: A panel to control the execution of three seplib programs. xtpanel-pipeline [NR]

Each parameter file dialog has a two buttons next to it. The lower button invokes a script that causes the program to self document. This documentation is shown in a text object. The script to do this is very short, note that the preprocessor is used to replace "PROG" by the appropriate program name. (The connection of stdin to /dev/tty is to force the program to self doc.)

```
button={ label=DONE action=QUIT }
text={ value=`( </dev/tty PROG 2>&1)` height=400 width=500}
```

Pressing the upper button brings up a subsidiary panel that contains interactive objects that can be used to specify the parameters. When the subsidiary panel is

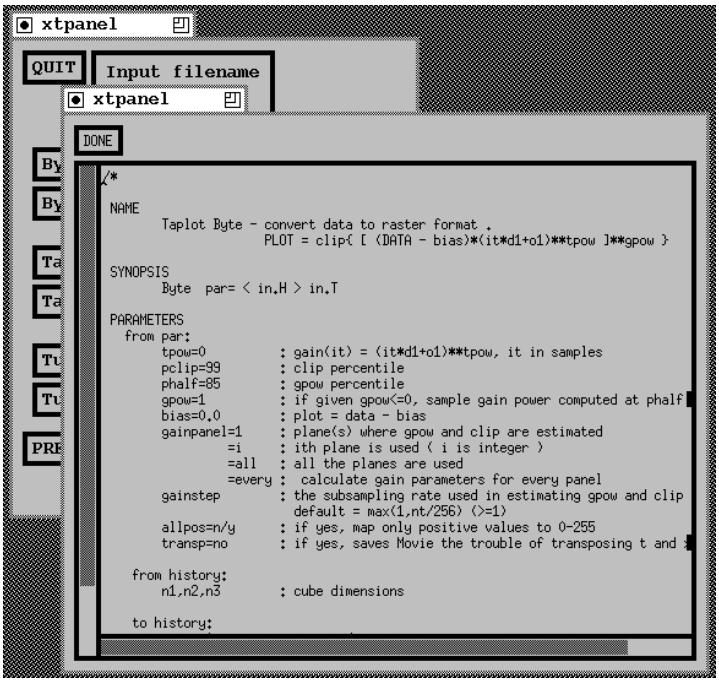

Figure 1.9: The documentation produced by pressing the Byte doc. button. xtpanel-pipedoc [NR]

closed the dialog is set to contain the parameters chosen on the subsidiary panel. Figure 1.10 shows the panel for Byte. The script file for this panel follows.

```
slider={ name=pclip min=50 max=100 value=98 format="%.1f" }

slider={ name=gpow min=0. max=4. value=1. format="%.2f" }

menubutton={ name=gainpanel label="gainpanel=>" value=1
             itemlist={ list="1 every all"}
           }

choice={ name=transp value=n
         item={ label=yes value=y }
         item={ label=no value=n  }
       }

button={ label=Done
 action="PRINT pclip=$pclip gpow=$gpow gainpanel=$gainpanel transp=$transp "
 action=QUIT
}
```

Finally Figure 1.11 shows the result of pressing the bottom button to run the command. Notice that the self doc is still available, as the xtpanel process to perform the self doc is run in the background. The final command is also run in the background so that you can have multiple results visible on the screen at the same time so that they can be compared.

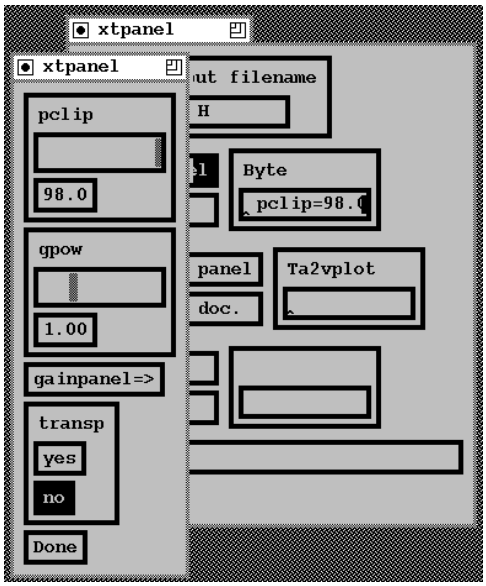

Figure 1.10: The panel for interactively setting Byte parameters is obtained by pressing the "Byte panel" button. xtpanel-bytepar [NR]

1.1.8 ed1D

Jon Claerbout (?) developed the interactive program `ed1D` to allow users to experiment with various aspects of one-dimensional seismology. Among other things, ed1D lets the user edit a function in the space or spatial frequency domain, and see the result in the other domain. In Figure 1.12 we implement this particular aspect of ed1D in an xtpanel. The script is 240 lines long, so we have not included it here. It is available on the CD-ROM version of this report.

1.1.9 Calculator

Figure 1.13 is an example that emphasizes xtpanel's ability to take advantage of the many powerful but non-interactive features built into UNIX. The `bc` calculator built into UNIX has most of the features of a standard scientific calculator. But it lacks a nice interactive interface. The X windows source distributed by MIT contains an interactive calculator `xcalc`. Written in C, xcalc is over 2000 lines long, and doesn't take advantage of the calculator already available in UNIX. The xtpanel script uses the bc calculator and provides an interface similar to that of xcalc. The script is 80 lines long, and available on the CD-ROM version of the report.

It is likely that this 80 line script required much less time to write than the xcalc application. And because the script syntax is very simple, adding features to this calculator would be a much easier task than adding features to xcalc.

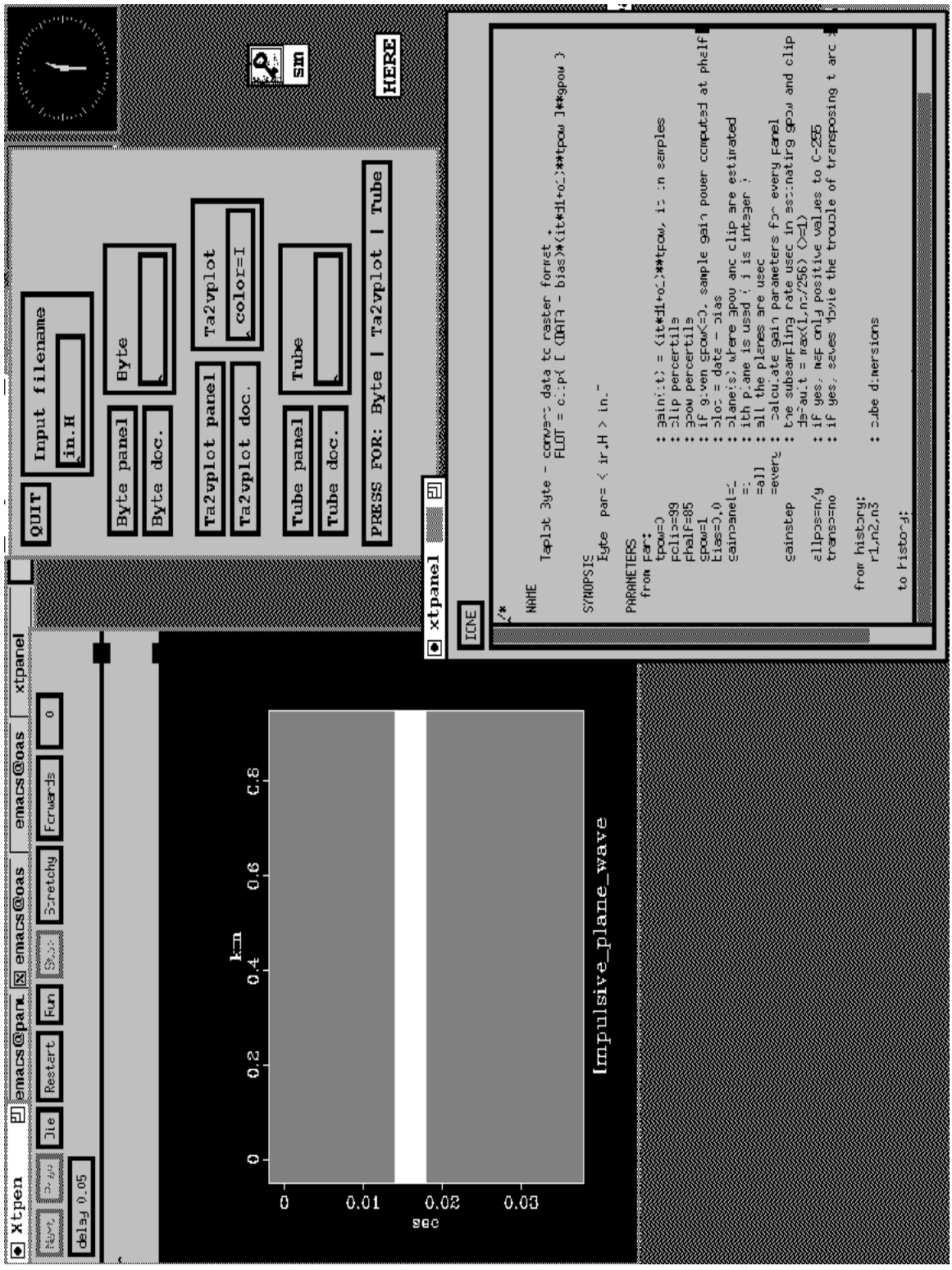

Figure 1.11: The result of pressing the action button. xtpanel-result [NR]

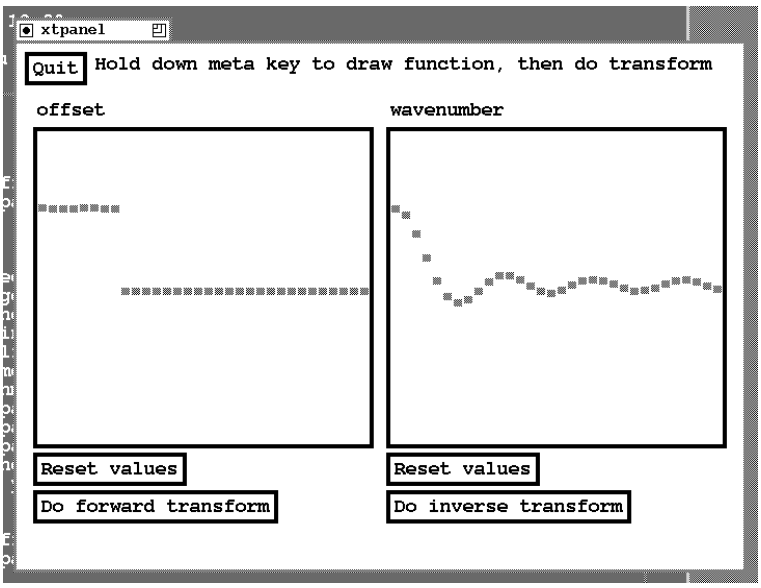

Figure 1.12: An xtpanel analog to Jon Claerbout's ed1D program, for experimenting with Fourier transforms. While this version is crude compared to the actual ed1D program, it took only about one hour to write. xtpanel-ed1D [NR]

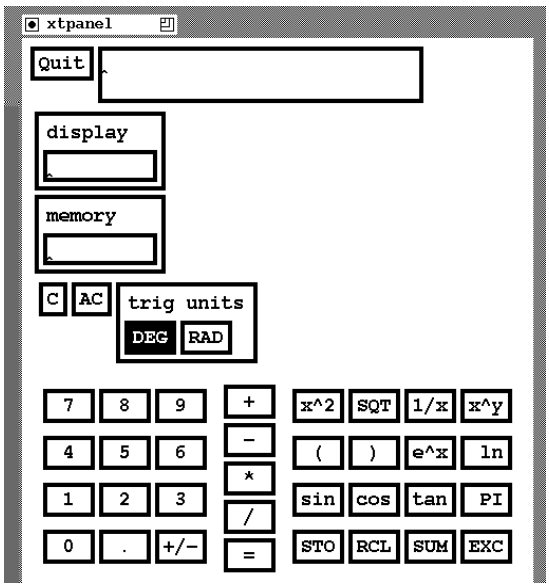

Figure 1.13: A calculator built using xtpanel. The 80 line script gives a calculator with features comparable to the 2000 line C program xcalc. xtpanel-calculator [NR]

1.1.10 The xtpanel generator

While the xtpanel script language is intended to be easy to read and use, it would be nice if it weren't necessary to learn it in order to use xtpanel. For this reason, we created the xtpanel generator. This is a series of xtpanels that let the user build a panel interactively, object by object. Figure 1.14 shows the top-level generator panel. Buttons are provided to add the various objects to the panel. At any point, the resulting script file can be examined, or previewed by running xtpanel on it.

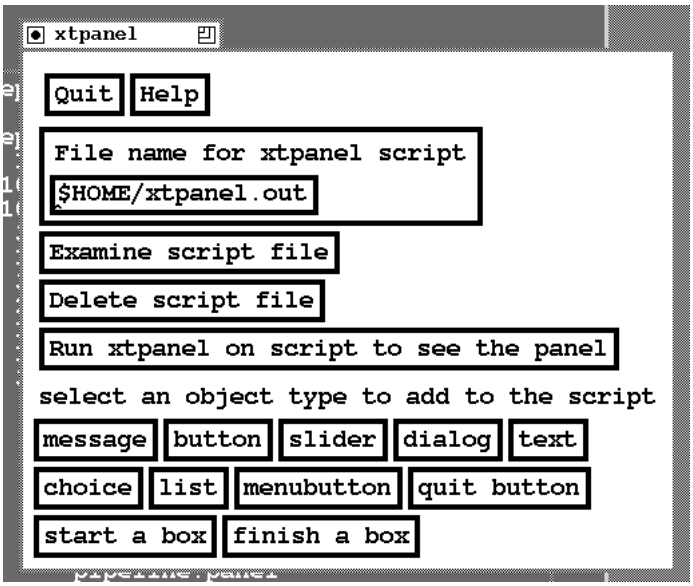

Figure 1.14: The xtpanel generator is a series of xtpanel scripts that let users build new xtpanels interactively. xtpanel-generator [NR]

1.1.11 Interactive help facility

Scrolling through a lengthy manual page to find a particular topic can be tedious. We created an interactive help facility for xtpanel, which is a series of panels that present, in a menu-driven form, the various parts of the manual page, along with some other pertinent information. The top level panel of the help facility is shown in Figure 1.15. General information is shown in the scrollable text field on the main panel; the menu lets the user choose from a list of additional topics.

1.2 Comparing xtpanel and other products

Many software packages address the general problem of building a graphical interface to simplify life for the end user. Xtpanel is worthwhile only if it offers something that these other packages (many of which are public domain) do not.

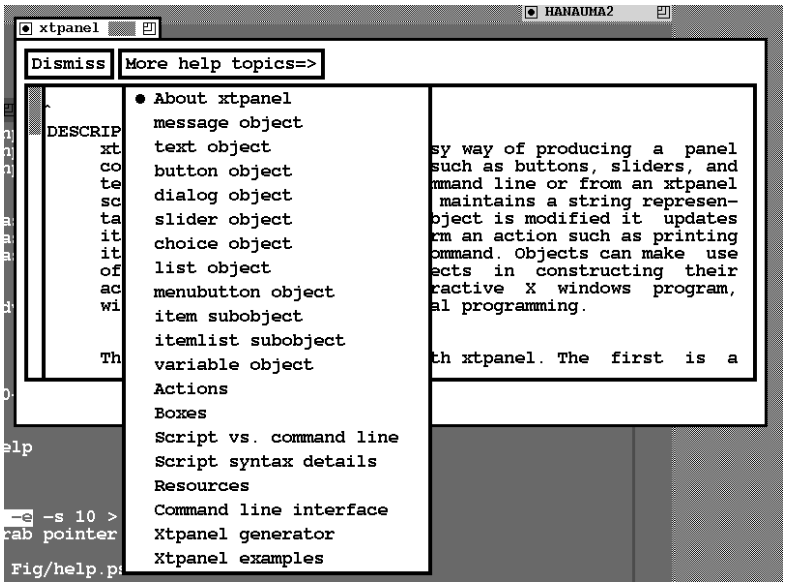

Figure 1.15: The xtpanel interactive help facility. The main panel displays general information, and a menu allows users to choose from a list of other topics. xtpanel-help [NR]

Some packages are complete graphical user interface builders. These programs let the user create an object, move it around on the screen or change its attributes – all interactively. While these programs are very powerful, they also tend to be somewhat difficult to use. Typically the work must be done interactively; the various configuration files are verbose and not easily edited by the non-expert. Such packages also are usually more self-contained; it is not as easy to integrate existing non-interactive UNIX software as it is in xtpanel.

Another set of tools takes a simpler approach, with all the configuration information specified on the command line. Two examples, distributed with the X windows source code from MIT, are `xmessage` and `xmenu`. These construct a panel with a set of messages, or a menu with several items. While such tools are very easy to use, they are typically quite limited in what they can do.

Xtpanel was designed to fill in the gap between these two classes of tools. It is meant to be very easy to use, but capable of generating panels that are complex and powerful. Also xtpanel takes more advantage than many other products of the power of UNIX, making it easy to run system commands and to incorporate their results into the action of the panel.

1.3 Interactivity with xtpanel

In Figure 1.5, xtpanel was used to create an interactive frontend to an existing seismic processing program, the bandpass filter program `Bandpass`. While building frontends in this way can be useful, often the overhead of having to re-run the entire job makes

this level of interactivity unsatisfying. Here we describe a way to use the interactivity of xtpanel from within programs.

Using xtpanel from within a seismic processing program offers some advantages. In the filtering example above, because the entire process is re-run each time a parameter is changed, a lot of extra work is done. Each time the `RUN` button is pressed, the input dataset (containing a single impulse) is recreated. Then the filter program must read in the data, as well as the parameters necessary to run the job. In such a trivial example, this extra work does not take much time. But in larger tasks, the extra overhead required to completely re-run a job may make the interactive performance poor.

To solve this problem, we devised a way to use xtpanel from within a processing program. After reading in its data and getting set up to run, a program brings up an xtpanel containing objects that specify some of the program parameters. Then the program waits for input from the xtpanel. When input is received, the program executes, and then waits for additional panel input. Additional panel input causes the program to be re-run, but without having to do all the overhead that was done at the start of the program.

The first program to use this interaction is a seismic data cube viewing program called `Cubeplot`. Cubeplot displays a perspective view of a seismic data cube. Displayed on the three faces of the cube are slices taken from within the data volume. In its standard batch mode of execution, the user specifies on the command line the three slices to be shown. If, on the command line, the user specifies `popup=y`, then a panel containing three sliders appears. As the user changes a slider, a different data slice is displayed on one of the three cube faces.

The xtpanel script is built into the C language source code for Cubeplot. Here is the script:

```
button={ label=Quit action="PRINTQUIT" action=QUIT }
vbox={
  hbox={ name=noborder width=400
    message={ value=axis-1 }
    scrollbar={ label="axis 1" name=pan1 min=0 max=N1 format="%.0f"
      width=300 value=FRAME1
      action="ASSIGN val1 $val"
      action="PRINT frame1=$(pan1) frame2=$(pan2) frame3=$(pan3)\n"
    }
    message={ name=val1 value=FRAME1  }
  }
  hbox={ name=noborder width=400
    message={ value=axis-2 }
    scrollbar={ label="axis 2" name=pan2 min=0 max=N2 format="%.0f"
      width=300 value=FRAME2
      action="ASSIGN val2 $val"
      action="PRINT frame1=$(pan1) frame2=$(pan2) frame3=$(pan3)\n"
    }
    message={ name=val2 value=FRAME2  }
  }
  hbox={ name=noborder width=400
    message={ value=axis-3 }
    scrollbar={ label="axis 3" name=pan3 min=0 max=N3 format="%.0f"
      width=300 value=FRAME3
      action="ASSIGN val3 $val"
      action="PRINT frame1=$(pan1) frame2=$(pan2) frame3=$(pan3)\n"
```

```
    }
    message={ name=val3 value=FRAME3  }
  }
}
```

However, the user can substitute a different script, containing whatever program parameters are of interest. Here are the instructions that appear in the on-line program documentation:

```
popup   Specifying popup=y brings up an xtpanel (if you have xtpanel
        installed) with three sliders. Moving these sliders changes
        the frames plotted on the three cube faces. If you pipe
        the output of Cubeplot to "Xtpen cachepipe=n" you will see
        the display update as the sliders are moved.
        You can specify your own xtpanel script file by doing
        popup_file=filename.
```

The program then calls two subroutines: `popup_start` brings up the xtpanel. This is called after the program's preliminary work (reading in the input data, getting parameter values, etc.) has been done. A second routine, `popup_check` is called after each pass through processing the data. This routine waits for output from the xtpanel. It then adds the new output to the table of parsed command line arguments. I.e. if the user moves a slider and the xtpanel prints "frame1=100" then after `popup_check` it is as though "frame1=100' had been specified on the command line. When this routine returns the program reprocesses the data (in the case of Cubeplot case plotting a new figure) using the new parameters.

Users interested in the details of these routines are referred to the CD-ROM version of this report, where the source code for Cubeplot and the popup routines is contained. The most important point to make about the source is that the modifications to the Cubeplot program were quite small — just two subroutine calls, and a few lines to interpret the new parameter values that come back from the panel. Interaction can easily be added to any program in this way.

Figure 1.16 displays a typical interactive session using the built-in xtpanel interaction of `Cubeplot`. If you are reading the report on CD-ROM, pressing the button at the end of the figure caption will bring up this example. Moving the sliders will change the display.

It is worth noting that although Cubeplot is a C language program, the xtpanel popup facility can just as easily be used from within Fortran and Ratfor programs.

Xtpanel future possibilities

We wrote xtpanel to fill a need, to add interactivity to the large library of non-interactive seismic processing programs available at SEP. In addition to making day-to-day data processing chores easier, this interactivity is also useful in building tutorials and interactive documents, such as this report. While other public-domain interface-building software exists, we believe that no other simple package is flexible enough to meet these needs.

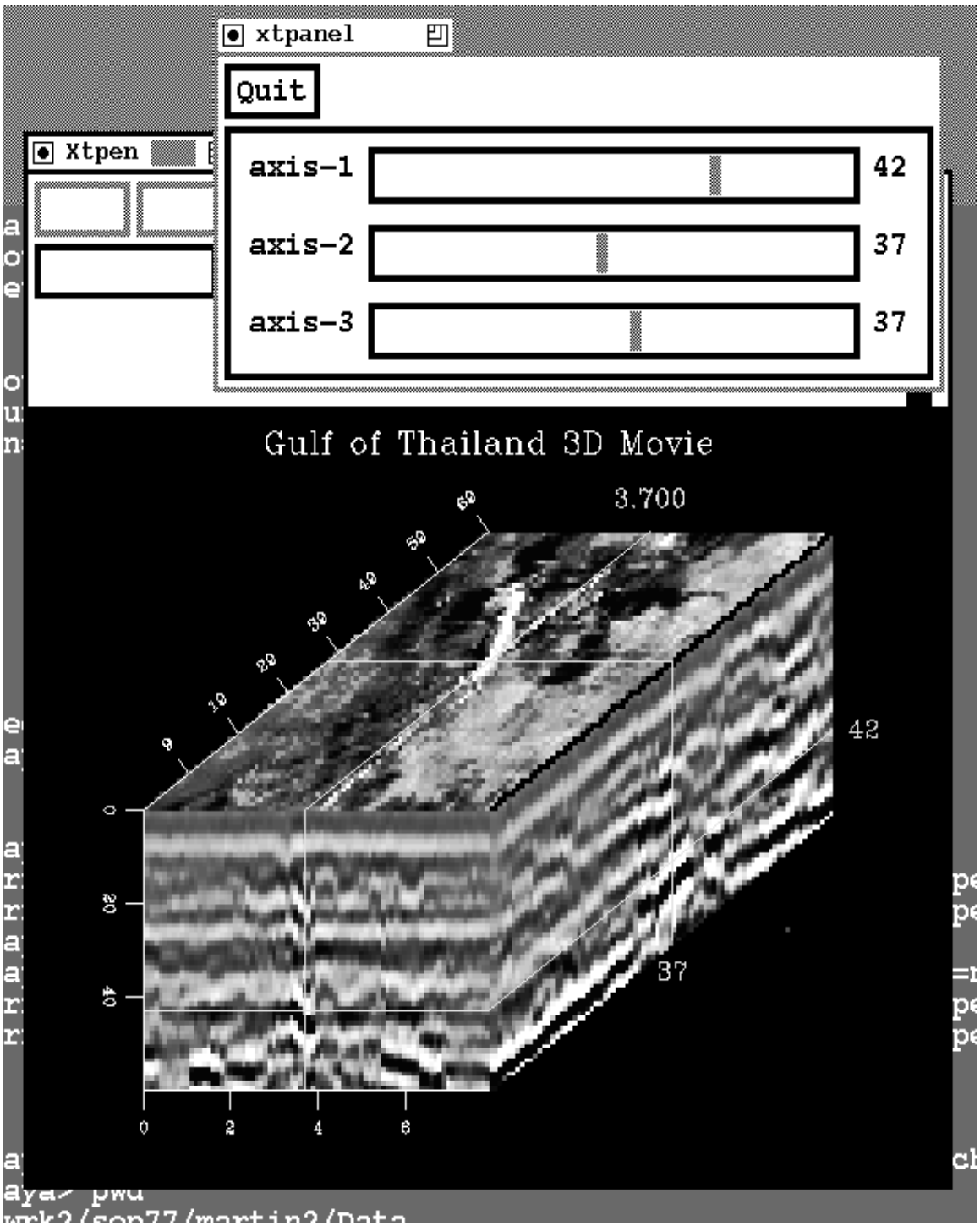

Figure 1.16: Interactive Cubeplot session using xtpanel. The xtpanel is brought up from within the program. Each time a slider is moved, a subroutine is run to re-draw the image. CD-ROM readers can click on the button at the end of this caption to run Cubeplot. xtpanel-cubeplot [NR]

Because it does not depend on other SEP software or geophysical software in general, xtpanel has been distributed through several public-domain source channels. Feedback from interested users has helped xtpanel to evolve to a reliable state in a short period of time.

There are several minor ways in which xtpanel could be improved in the future. Support of Motif in addition to the MIT Athena widget set would make xtpanel compatible with the software environments of more potential users. Because of the modular way in which xtpanel has been written, and because the Athena widgets and Motif have many features in common, adding Motif support would not be very difficult.

Another promising possibility is to be able to put images (pixmaps) in the background of xtpanel objects. If this were possible, one could, for example, very quickly build seismic tools that incorporate data picking, by having an image of the data in the background of certain interactive objects.