# Xtpanel: an interactive panel builder

*Steve Cole & Dave Nichols* [1]

/usr/local/src/our/word/tutorials/xtpanel

## ABSTRACT

Xtpanel builds interactive panels — containing objects such as buttons, sliders, and dialog boxes — from a simple script file or from the command line. With xtpanel, one can quickly add a layer of interactivity to existing software. For instance, a velocity analysis tool can be created in minutes by using xtpanel to add an interactive front end, where sliders choose velocities from a user-specified range, to an existing NMO correction program. Panels can take advantage not only of existing application programs, but also the utilities built into the UNIX operating system. In one example, we use an 80-line script to build an X windows calculator by making a front end to the line-oriented UNIX calculator `bc`. Another utility built with xtpanel is the xtpanel generator. This is a series of panels that let users build new panels interactively, without having to learn and use the script language.

## INTRODUCTION

For several years, members of SEP have looked for a way to add interactivity to existing UNIX programs. The most ambitious project, `SepView` (Dulac, 1988), was a complete graphical interface environment and interface generator. SepView was not completed at SEP, where the necessary development time and expertise were not available. However, the work has been completed elsewhere.

Two simpler schemes were developed later — `IPE` (Cole, 1990a) and `xvpanel` (Cole, 1990b).

IPE (an acronym for Interactive Processing Environment) had two serious limitations:

[1]**email:** Author has no known email address.

- It was a two-step process: a code generator, `Ipemake`, took in a simple ASCII description of a program's parameters, and generated C language code that had to be compiled into IPE. A new program was generated for each new interface added.
- IPE was tied to the pipeline processing model, after parameters were set the end result was always a simple pipeline of processes to be run.

Xvpanel was an improvement, in that the intermediate step of generating and compiling C code was eliminated. However, several problems remained:

- Written using the XView toolkit developed by Sun Microsystems, which is not very portable.
- The command line interface was unfriendly. All parameters had to be specified, in a particular order, or xvpanel would fail to work.
- Like IPE, still tied to the pipeline processing model.

Working from the base of experience gained in developing these earlier schemes we have written a new program, called `xtpanel`, that is more powerful, more flexible, and easier to use than these earlier programs. Xtpanel has none of the disadvantages of the earlier products, and has several advantages:

- Written using the X toolkit, included with all versions of X windows. Very portable.
- Has an easy to read, easy to use script language in which panels can be built.
- Not tied to the pipeline processing model. Panel objects can act independently of one another, or in concert.
- Comes with the xtpanel generator, a series of xtpanel scripts that guide a user through the process of building an xtpanel interactively.
- Comes with an interactive tour of some examples, as well as an interactive help facility.

While xtpanel only recently reached its present state, there are already a number of interesting geophysical projects that are using xtpanel. For example, Stan Ruppert, a Ph.D. student in Geophysics at Stanford, is modeling the shear wave velocity structure of the crust and upper mantle beneath the Colorado Plateau. Stan generates synthetic seismograms given a velocity model, and compares them against recorded seismograms. What was a tedious process, where Stan manually edited a velocity parameter file, then ran a batch program to generate the synthetic seismogram, was made interactive using xtpanel in the course of a few hours. Matthias Schwab, in this report (?), uses xtpanel to build a tutorial on wavelet transforms.

In addition to these specific projects, xtpanel is also used by several authors in this report to provide interactivity. Those reading the CD-ROM version of this report are able to click on buttons attached to most figures. In many cases, this will bring up an xtpanel that drives an interactive reproduction of the figure, perhaps giving the reader an opportunity to change some of the relevant parameters.

Later in this report, we illustrate the use of xtpanel with several examples. First, we describe xtpanel and its features in greater detail.

## INTERACTIVE OBJECTS

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)

### 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.

### Object names

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

### 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.

## 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.

### 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 shows the panel produced.

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


Figure 1: A simple panel with two buttons. buttons [NR]

### Simple menu

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

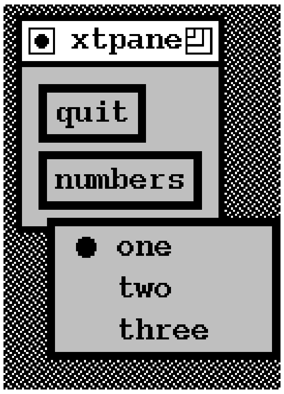



Figure 2: A simple panel with a menu and a quit button. menubutton [NR]

### Simple interactive parameter selection

This exmaple shows how xtpanel can be used in conjunction with the cake rules for an interactive document (Claerbout and Nichols, 1990) 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 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 " }
```

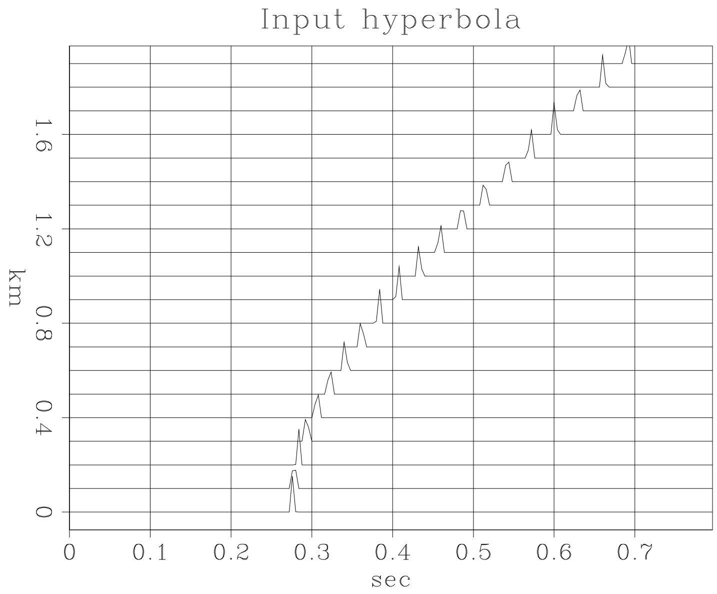



Figure 3: A synthetic dataset, click on the button to choose the correct NMO velocity. nmo [ER]

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 4 shows the interactive panel in use.

### 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 }
```

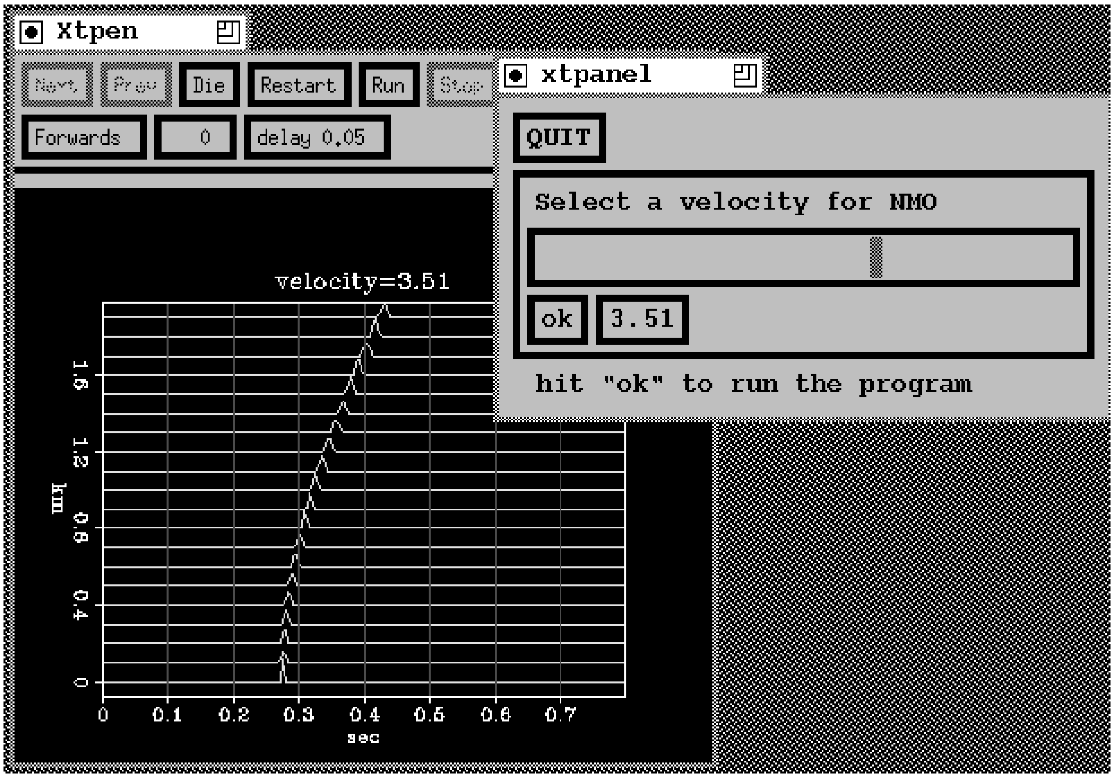



Figure 4: The simple velocity selection panel in use. nmores [NR]

```
button={ label=GO
  action="Bandpass flo=$flo fhi=$fhi phase=$phase < $file \
            > /tmp/band.h out=/tmp/_band.h@; \
          Byte pclip=$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" }
  }
}
```

## 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.

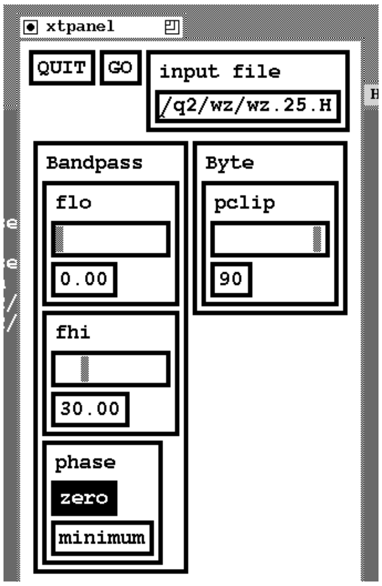



Figure 5: A panel to control the execution of two seplib programs. simplepipe [NR]

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 }
text={ name=display height=300 width=500 }
var={ name=wind value=" Window n3=1 " }
var={ name=disfil value=" Disfil number=n " }
var={ name=edit value=" sed -e 's,n2=,,g' " }
var={ name=killcr value=" tr -d '\012' " }
var={ name=cols }
var={ name=file value=FILE }
slider={ name=slice label="select plane to view" value=0 min=0
  max=`Get n3 < FILE | sed -e 's,n3=,,g' ` format="%.0f" width=300
  action="ASSIGN cols `Get n2 < $file | $edit | $killcr `"
  action="ASSIGN display `$wind f3=$slice < $file | $disfil col=$cols `"
}
```

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.

### 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

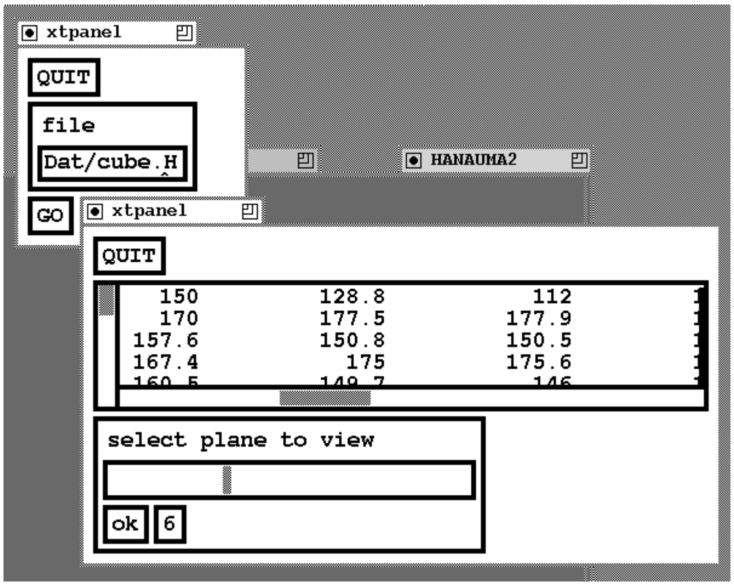



Figure 6: A viewer to examine SEP data cubes. viewer [NR]

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 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=" " }
}
```

### 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 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.

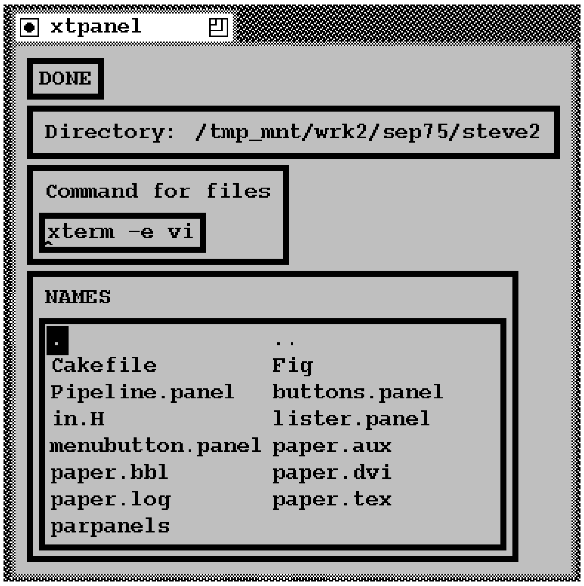



Figure 7: A panel to traverse a Unix filesystem. lister [NR]

```
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 "
}
```

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 }
```

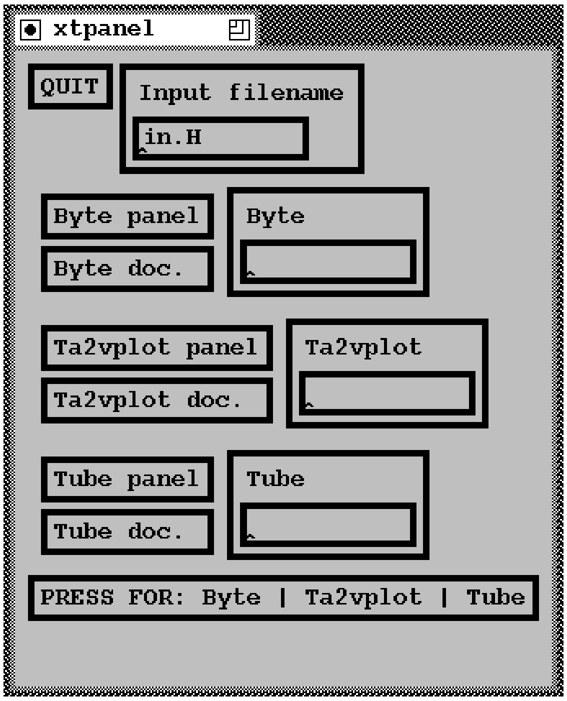



Figure 8: A panel to control the execution of three seplib programs. pipeline [NR]

```
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 closed the dialog is set to contain the parameters chosen on the subsidiary panel. Figure 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"}
           }

hbox={ message={ value=transp  }
       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 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

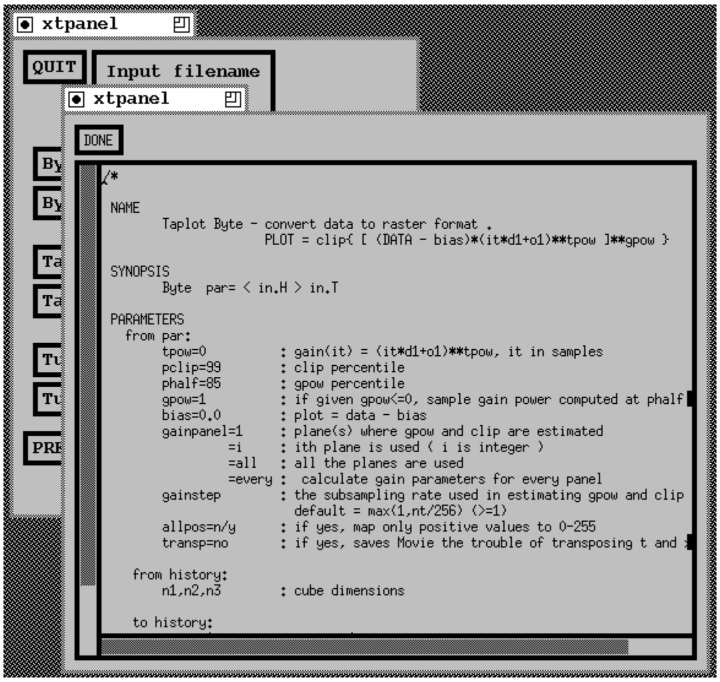
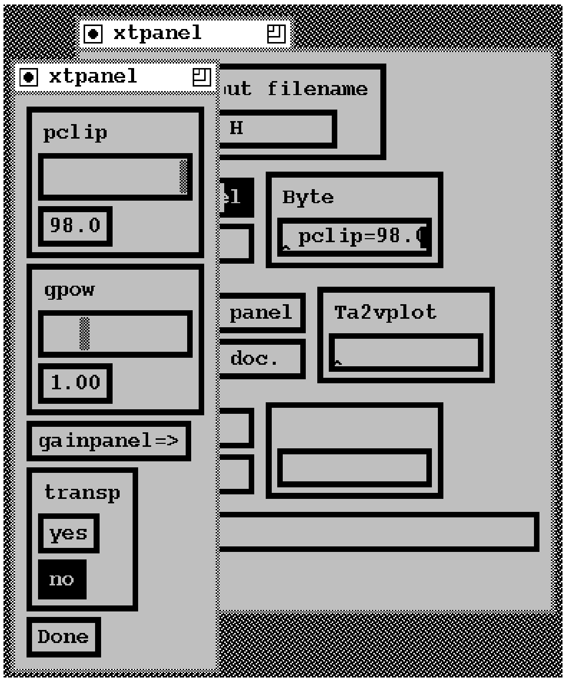



Figure 9: The documentation produced by pressing the Byte doc. button. pipedoc [NR]

Figure 10: The panel for interactively setting Byte parameters is obtained by pressing the "Byte panel" button. bytepar [NR]



so that you can have multiple results visible on the screen at the same time so that they can be compared.

### ed1D

Jon Claerbout (Claerbout, 1991) 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 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.

### Calculator

Figure 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.

### 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 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.

### 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 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.

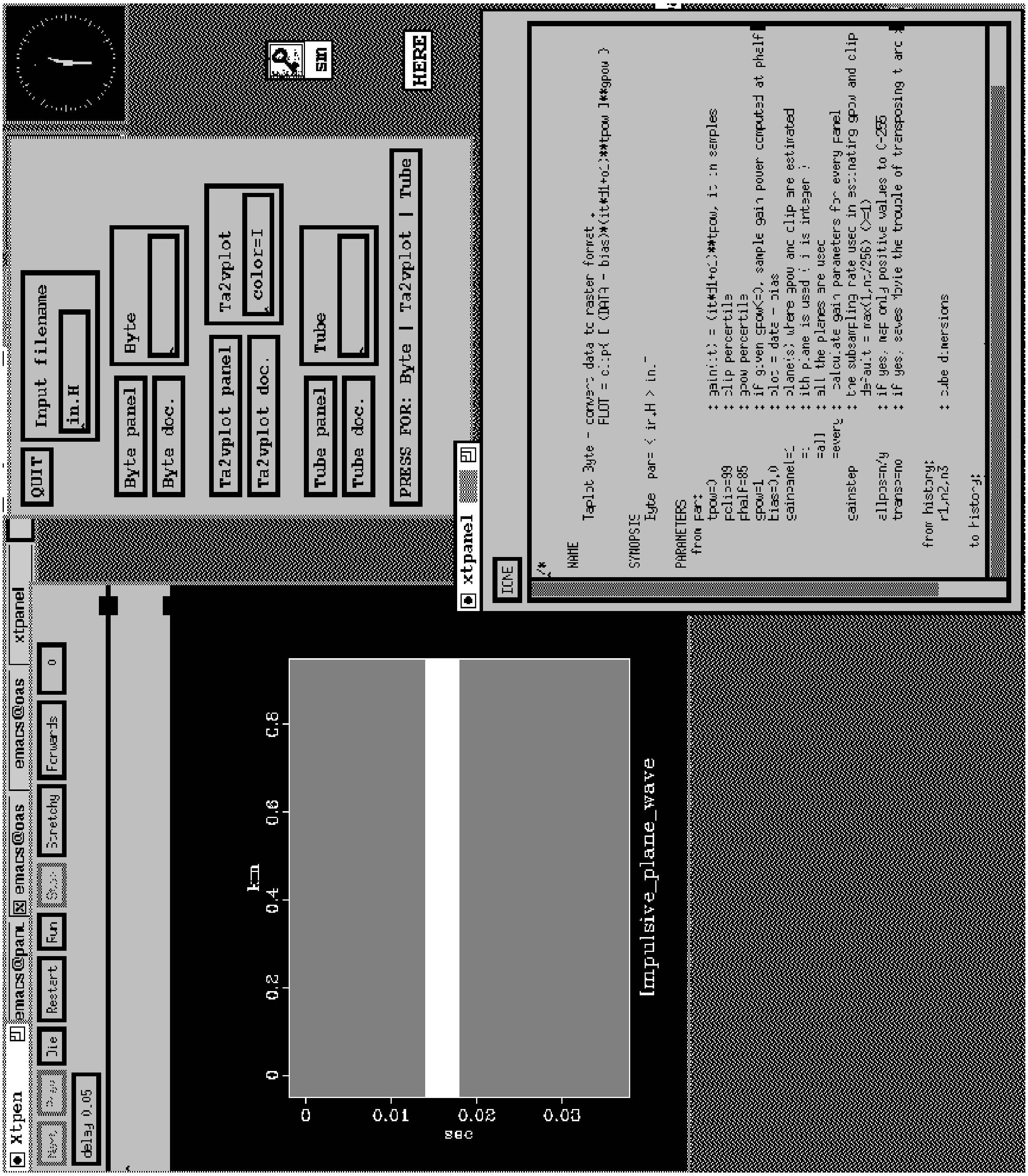


Figure 11: The result of pressing the action button. result [NR]

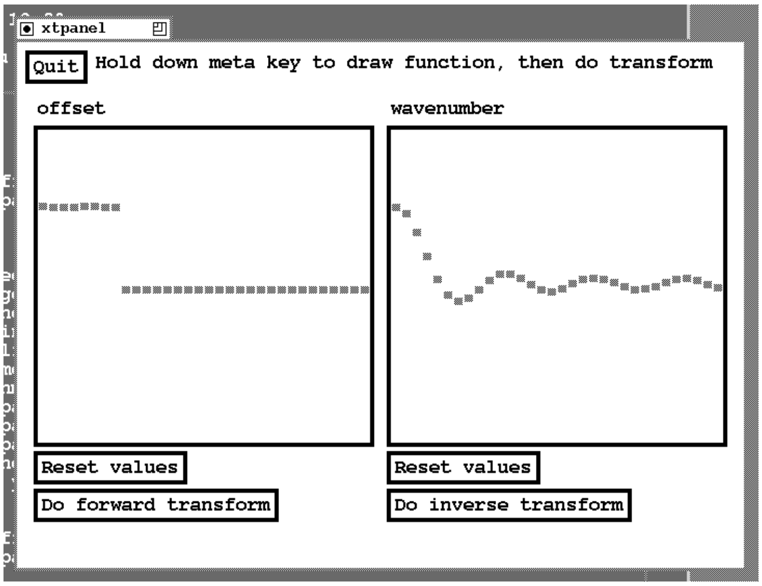



Figure 12: An xtpanel analog to Jon Claerbout's ed1D program, for experimenting with Fourier transforms. While this version is crude compared to the actual ed1D program, it took only about one hour to write. ed1D [NR]

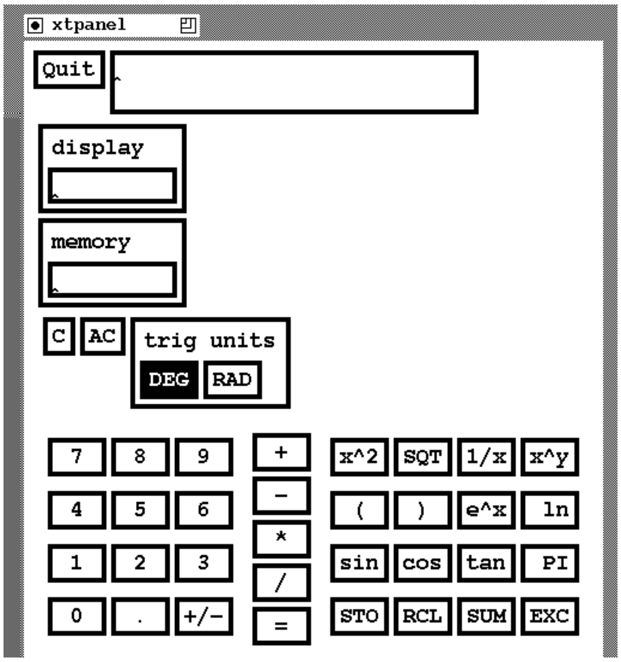



Figure 13: A calculator built using xtpanel. The 80 line script gives a calculator with features comparable to the 2000 line C program xcalc. calculator [NR]

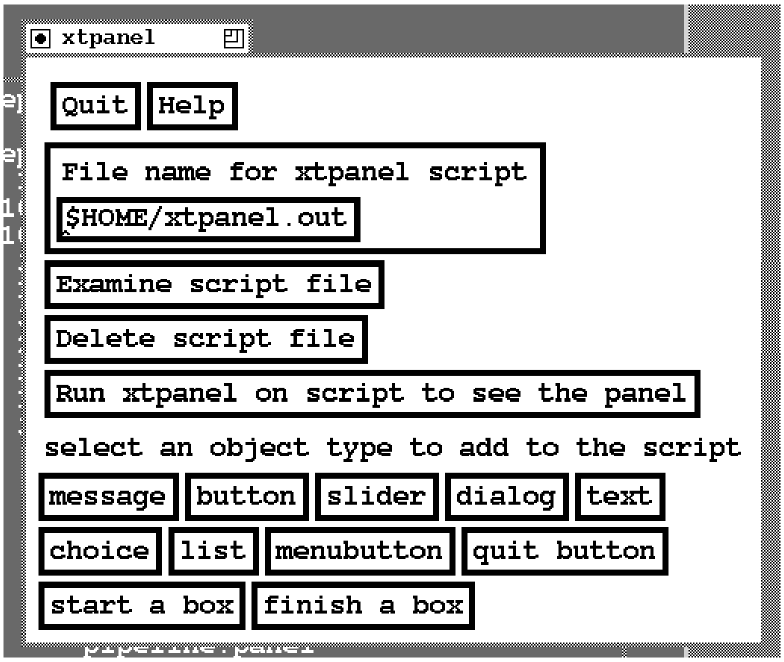



Figure 14: The xtpanel generator is a series of xtpanel scripts that let users build new xtpanels interactively. generator [NR]

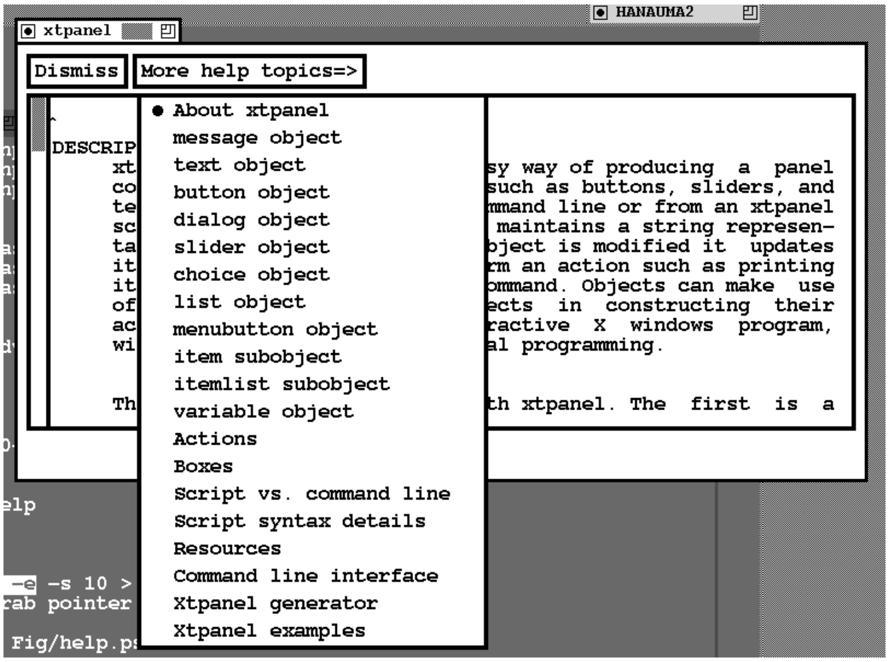



Figure 15: The xtpanel interactive help facility. The main panel displays general information, and a menu allows users to choose from a list of other topics. help [NR]

## 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.

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.

## CONCLUSIONS

Interactive software is time consuming to write. Xtpanel aims to provide a way to make existing batch software interactive, and a way to build new interactive programs using a simple script language instead of having to write X windows software. Xtpanel's primary strengths are its flexibility, which allows easy access to the powerful tools built into UNIX, and its ease of use. We hope that this combination of flexibility and ease of use will allow xtpanel to be applicable to a wide range of problems.

## REFERENCES

Claerbout, J. F., and Nichols, D., 1990, Why active documents need cake: SEP–**67**, 145–148.

Claerbout, J. F., 1991, Interactive one dimensional seismology program ed1D: SEP–**71**, 293–294.

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

Cole, S., 1990b, Xvpanel : Interactive programs in one line or less: SEP–**67**, 161–166.

Dulac, J. C., 1988, A user interface manager : SepView: SEP–**59**, 223–240.

## APPENDIX A

The on-line manual page for xtpanel follows. If you are viewing the document on CD-ROM, the manual page will not appear here, but it is installed on the CD, and you can simply type `man xtpanel` to see it. Or you can press the button in Figure 15 and use the interactive help facility.