XScript – Shell Programming with X

Jan Newmarch
Faculty of Information Science and Technology
University of Canberra
PO Box 1, Belconnen,
ACT 2616, Australia
email:jan@ise.canberra.edu.au

Abstract

Shell programs are still widely used in Unix systems. This paper reports on a system that supplies a number of X Window applications that are suitable for use in shell scripts. These can be used in any shell scripts, including pipelines and command substitution. The applications include graphical versions of typical character–based commands and also a number which are specific to X. This allows shell programmers to write applications using the X Window System without having to learn to program in X.

This paper is due to appear in the X Resource sometime in 1995.

1. Introduction

The X Window System is now pervasive on workstations, with an increasingly large number of X applications available. Typically such applications are built using a toolkit such as Motif or Interviews, in a procedural language such as C or C++.

Programming at this level is complex and time–consuming, with even simple applications requiring a steep learning curve to write. There are a variety of alternative approaches, each attempting to simplify this, such as: alternative language bindings to the toolkits, custom languages for special situations, GUI builders of various levels, etc.

This paper reports on a project that attempts to exploit a niche in the spectrum of approaches by leveraging from an existing base of shell programmers. It does so by creating a set of applications that run as X Window programs, but which do not require the ability to write X Window code, and do not require learning a new language in order to use them. For example, display of prompts and messages of various kinds (error messages, warnings), and filters that display information graphically and allow selections to be passed to stdout.

This approach relies on finding a rich enough set of applications that can be used in this role. A set of applications that are useful is described here, and a number of examples are given of their use. However, the set is extensible, so that an X programmer can extend this set for a non–X programmer to use (much as a C programmer can extend the set of character–based applications for use by a shell programmer or casual Unix user).

The next section considers some of the programming alternatives that are available in the X System, and describes where this proposal fits in. The next sections look at the set of applications that form the core of this system. Following this, some options for implementation are considered. Some of the programs are designed to be used in shell pipelines; this covers a variety of uses, but is not of sufficient generality as a communication method to handle many cases. A section discusses the ''send'' protocol from Tk, and how this is used here to extend the range of uses of this system. In addition mechanisms for making drag and drop, and selections available to the shell programmer are given, along with examples of their use. Finally, some conclusions are offered.

2. Levels of programming

The most common method of building X Window applications is to use a standard programming language such as C or C++. Over the years, the range of languages bound to each toolkit has increased. For example, for Motif there are bindings from C, C++, Ada, Lisp, Prolog, Korn Shell and tcl [9]. For the Tk toolkit there are bindings from tcl, Scheme, Perl and Python [12]. Each of these has its adherents, for a variety of reasons.

Some of these bindings offer the virtue of being high–level command languages, with sufficient power and simplicity that they make the task of X Window programming much simpler, without losing too much expressibility. Systems such as tcl/Tk [8], tclMotif [5], WAFE [4] and the DTKSH [13] typically claim a code saving of upto 90% compared to the corresponding C code.

Despite this, the fact remains that in each of these the programmer has to deal directly with the objects of the toolkit such as PushButtons, Lists, etc. In addition, they will have to be familiar with the callback model, and how such things as timer functions have to interact with the X event loop processing. This is not to denigrate such bindings, but only to note that they still involve X programming.

Systems such as WCL [10] and Xtent [1] adopt an approach which simplifies the GUI side by deriving the widget structure from a declarative approach based on the X resource mechanism. This still requires a considerable knowledge of X.

A custom language like xgen [2] may be used in situations that require interaction between a set of widgets and external applications, such as pressing a button triggering a particular application. However, this system does not possess full language capability, and is an additional layer of learning. Within its domain, though, it does offer a simple way of building applications with a GUI interface.

3. Shell programming with Applets

Due to their prevalence as Unix command languages, there is a large number of programmers who are familiar with programming in the Bourne and C shells [11]. These programmers are used to using pipelines, I/O redirection and command substitution (where output from the command is left in place, such as ‘date‘) to link together applications from a variety of sources to build more complex systems.

This relies on a design philosophy of Unix tools, that they should be small applications designed to do one job well, and to be able to be coupled together. In the PC world, which has recently discovered the benefits of such an approach, many Unix tools would be described as “applets” [14].

This paper describes a set of “applets” which are designed to fit in with this philosophy, but which are distinguished by being X Window applications. That is, they can be used in pipelines, I/O redirection and command substitution. In this way they can be used by the shell programmer to utilise the X Window system without requiring the user to directly program (or even to know how to) in X. The current set of applets are summarised in the Table. More details, and examples of use, are given in the following sections.

xask	ask a question and return an exit code
xdrop	send a drop or paste to standard output
xerror	post an error message
xfileSelect	select a file from a list and write to standard output
xinfo	post an information message
xiselect	select from a list and write to standard output
xprompt	prompt for an answer
xselect	select from a list and write to standard output
xsend	send a command to an application
xwarn	post a warning message

Information applets

The Unix command `echo` is used to show a string, or `cat` to display file contents. A graphical environment is able to attach more meaning to a message than simply displaying it. The Motif toolkit [3] supplies a number of information widgets, and it is a simple manner to package them into applets. These typically display text with pixmaps to show the type of information displayed – information, error, or warning.

The primary set of information applets in XScript are `xinfo` to display a piece of text with a big “i” to denote information content, `xerror` to inform that an error has occurred, and `xwarn` to show that a serious condition has arisen.

For example,

```
xinfo Todays date is `date`
```

is shown in Figure 1.

A common beginning to any shell script is a check on the number of arguments, with an error message if the number is wrong. A graphical version of such a check is

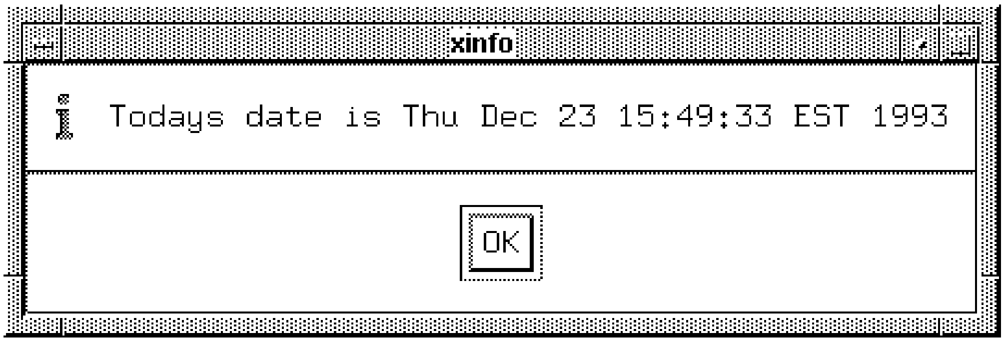

Figure 1: xinfo

```
if [ $# -lt 1 ]
then
    xerror "Usage: $0 arg args"
    exit 1
fi
```

which displays the error in Figure 2.

Figure 2: xerror

Query applets

Many interactive scripts requires information from the user. This may range from simple text entry ("What is your age?") to selection from information that can be supplied from the script itself or from the system.

The simplest applet is one that asks a question and expects a yes/no answer. The command `xask` displays its command line arguments in a question box, with a Yes and a No button.This returns a Boolean value in a similar way to the Unix command `expr`: selection of one of these buttons returns an appropriate exit code for shell programs: zero for Yes, one for No, while it also writes the value to stdout of one for Yes, zero for No (the inconsistency in values mirrors that of the command `expr`).

For example,

```
if xask Remove all files? > /dev/null
then
```

```
        rm -rf *
    fi
```

This is shown in Figure 3.

Figure 3: xask

An option to `xask` allows it to be "persistent" so that it can be used as the source of a stream of values. An example using this is shown later.

More complex query commands require a textual response. The command `xprompt` displays a piece of text and an edit region to enter the response. When the OK button is pressed,or the return key pressed, the text entered is sent to standard output, as in

```
    name=`xprompt What is your name?`
```

`xprompt` also allows a default value to be specified, in selected form. This means that if the default is ok, a simple return or click on OK will send this value to standard output. If the default is not wanted, entering any text will replace this selected default value. For example,

```
    name=`xprompt -text "John Smith" \
               What is your name?`
```

is shown in Figure 4.

`xfileSelect` allows the user to select a file name from a list of files. Options allow the directory and the file selection mask to be set. See Figure 5

Pipeline applets

Unix has achieved a large measure of its success due to the ability of the shells to connect applets in pipelines. An applet in a pipeline receives an unspecified amount of input and produces output according to its function. This is quite open–ended, and many of the Unix tools are set up so that they can act as filters in pipelines.

To use graphical applets in a pipeline requires that they must take their input and display it in a suitable graphic format. The applet can then either select or filter the input and pass suitable components to standard output.

XScript supplies two pipeline applets, both forms of lists. The first, `xselect`, shows a list of lines received from standard input. `xselect` will display lines as re-

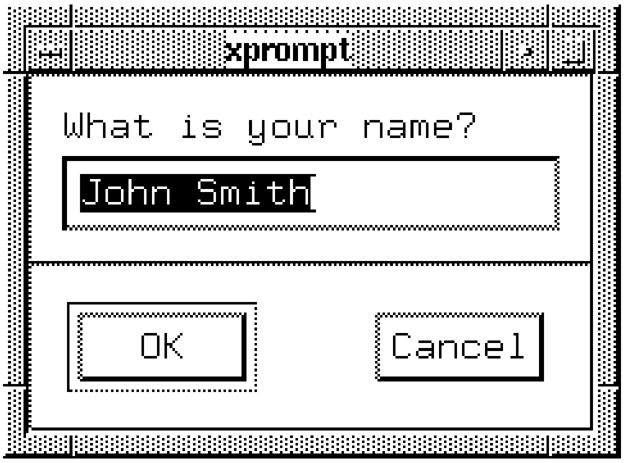

Figure 4: xprompt

ceived, so that even in a pipeline that is producing results slowly, something can still be shown. The simplest form is simply as the terminating end of a pipeline, to display the data, as in

```
ls | xselect -message "File list:"
```

shown in Figure 6.

`xselect` can also be used as a filter in a pipeline. When the list is in single–select mode, pressing on an item sends the item to standard output, newline terminated. For example, interactive file removal can be performed by

```
rm `ls | xselect -message \
        "Click on files to remove"
```

More complex scripts can be used, such as the following which is a simple file browser:

```
ls | xselect |
while read file
do
    xedit $file &
done
```

where each selected file is passed to the editor `xedit` for display.

A similar use to this is as a "sounds" browser. This can be done by simply running the above script in a directory containing audio files, where instead of `xedit` a sound player is used. A more sophisticated script could examine the file type and choose the appropriate application.

Other list selection models include multiple select, extended selection and browsing. Options to `xselect` allow these alternative modes to be set. In these models an extra button called "Send" is added, and Items selected using these models are only sent to standard output when the "Send" button is pressed. See Figure 7

The other applet, `xiselect` (for x–image–select), is similar to `xselect`, but allows further graphical information to be displayed. It takes as input lines that consist

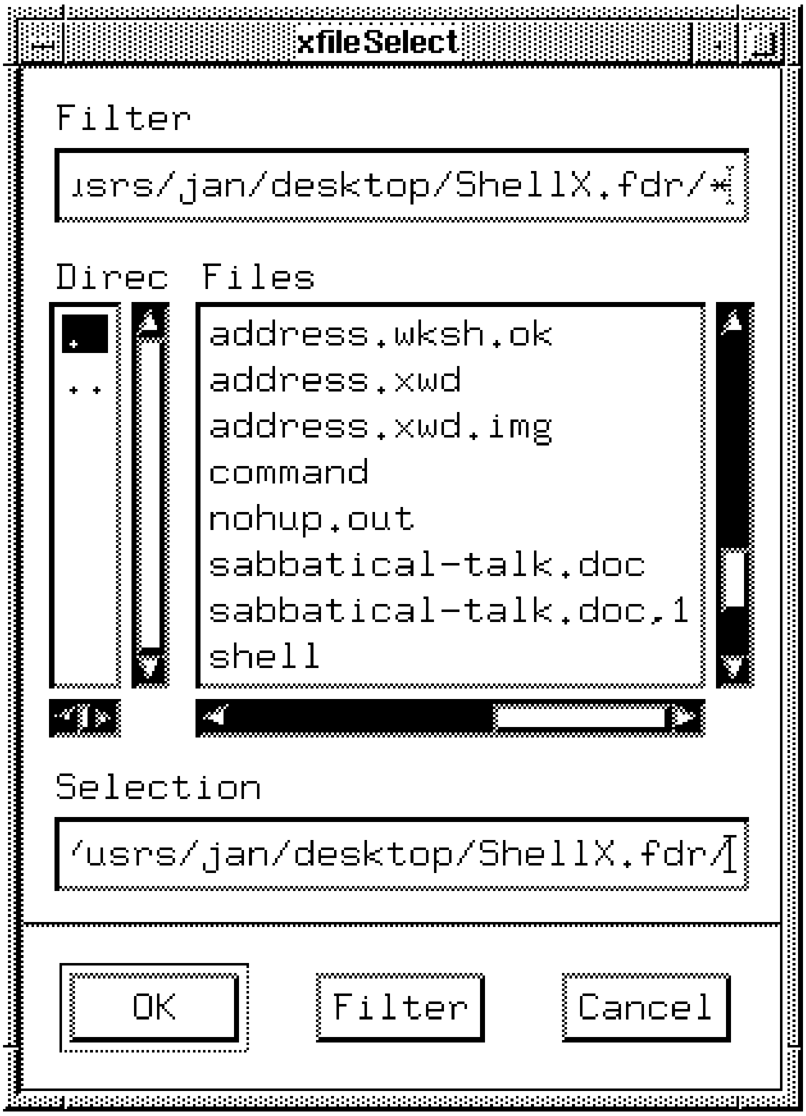

Figure 5: xfileSelect

of the name of a bitmap file and the item to display. This bitmap file is searched for in a large set of directories which is quite customisable. For example, a simple list of users logged in can be shown by

```
who | xselect
```

A more informative display can be given by passing the output from `who` through a filter that prepends each line with the bitmap image of each user. This filter could be a shell script, an awk script, or anything else that could perform a trivial database lookup. This could result in a script

```
who | images | xiselect
```

This is shown in Figure 8.

Providing Help

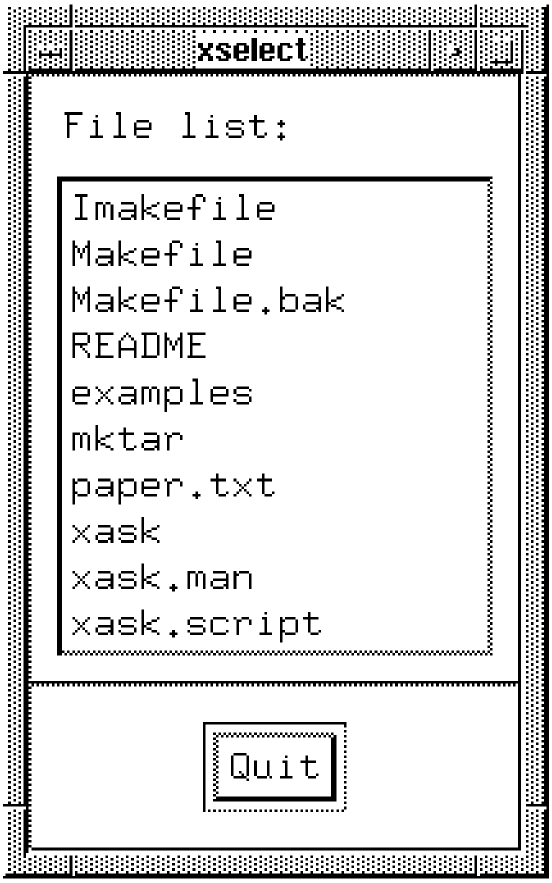

Figure 5: xselect

The majority of applets will be simple and require no further explanation. However it is inevitable that some will cover complex events, and for these some help may be required. Help is generally viewed as application–wide or context sensitive. For these applets, there is really no context apart from the application itself, so complex mechanisms are hardly required.

Each applet has command line options to display help with content either from a string or from a file. If one of these options is used, an extra Help button is added to the applet to bring up a Help dialog with the specified text. For example

```
xask remove all files? \
    -help "This will run rm -rf *"
```

This is shown in Figure 9

Timing Out

It is the norm that applications make too many assumptions about user behaviour, or make too few. A typical example of one that makes too few for me is an editor that, when you ask it to print a file, puts up a dialog whose primary purpose is to ask how many copies to print. To me it is obvious: one only, unless I ask for a special print command. I have spent quite a few hours in total waiting at the printer while the computer is patiently waiting for me to complete the ‘‘number of copies’’ dialog, as I have gone immediately to the printer, forgetting that the dialog was going to be posted.

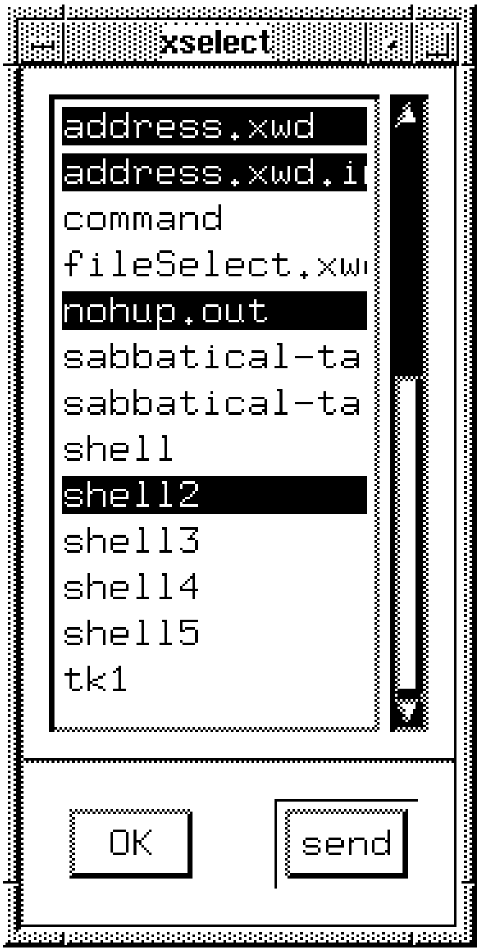

Figure 4: multiple select mode

Either there should be a separate command for special cases, or there should be a mechanism for supplying a default after a reasonable time interval has passed.

For this reason, most of these applets have a timeout option, which will return a special exit code of 2 if the timeout occurs. This allows shell scripts such as

```
copies=`xprompt -timeout 60 How many copies?`
if [ $? -eq 2 ]
then
    copies=1
fi
```

Sending Messages

The tcl/Tk toolkit has a comand called "send" [8] which acts as a communication mechanism between X applications, based on server properties. This allows one application to send a command in the language tcl to another application that is using a tcl interpreter. In the tcl/Tk system this obviously occurs since the whole system is an extension to the basic tcl interpreter.

This allows one application to send arbitrary tcl commands to another. To be useful, this usually requires the sending application to know enough of the structure of the

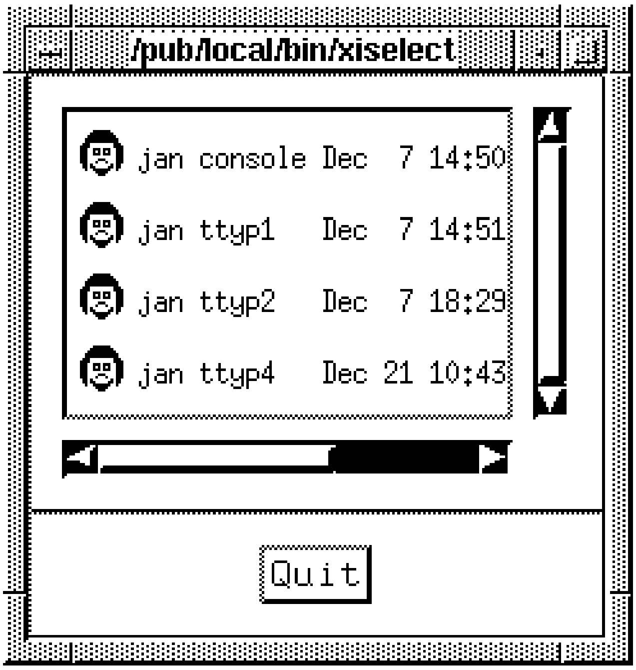

Figure 6: xiselect

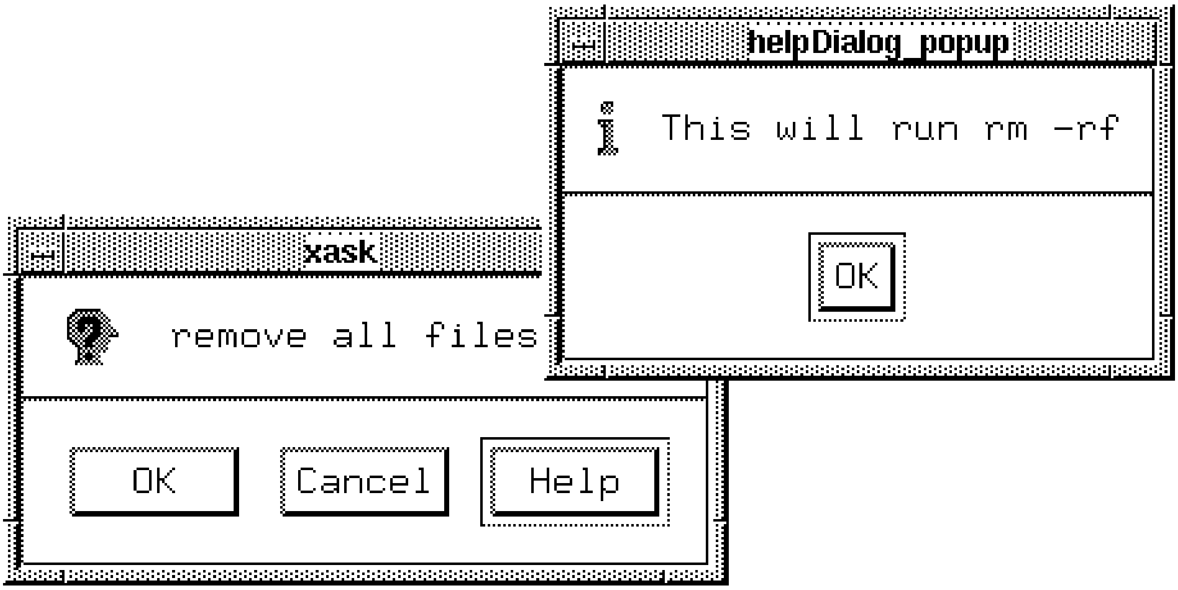

Figure 7: help

other to know what would be useful to get the other to do. A simple example is a debugger sending instructions to an editor to display the current line of execution.

The author has extracted ‘‘send’’ from Tk so that it can be linked in as library to Xt applications [7]. This has been explored as a preliminary to a scripting language for Xt applications [6].

The applets in XScript are all "send aware" in that each contains a tcl interpreter that is able to respond to commands sent to it. However, the intention is not to get users of this system to learn tcl as well. To avoid this, each applet has a documented set of commands that can be sent to it (in addition, it will respond to many more, but they are not documented). For example, the applets that display a string (such as `xask`) can be sent the command

```
setMessage text
```

An additional applet is `xsend`. This is just to send a tcl command to an applet from a shell program.

The principal use of this mechanism is to overcome limitations of the shell pipeline as a means of communication. For example, consider the GUI equivalent of `rm -i`:

```
for i in $files
do
    if xask remove $i?
    then
         rm $i
    fi
done
```

Each time around the loop this will create another instance of `xask`. This can be quite exasperating in some circumstances such as when the window manager is set up to interactively place each window. In such a case the user will first have to place and only then answer the question for each file. In such a case, it would be far preferable for the applet to be "persistent" and to update the question each time round the loop. This can be done using `send`, using the output from `xask` to stdout:

```
xask -persistent |
for i in $files
do
    xsend xask setMessage remove $i?
    read ans
    if [ $ans -eq 0 ]
    then
        rm $i
    fi
done
xsend xask exit
```

This example is partially resilient to errors. If `xask` were destroyed, then `xsend` will timeout (under the current implementation) since it cannot deliver its message. However, the `read` will fill `ans` with a null string and this will cause a subsequent parse error in the `test` statement. To guard against this, the `test` should really be

```
[ -n "$ans" -a "$ans" -eq 0 ]
```

In a similar manner `xinfo` can be turned into a clock, updated each second:

```
xinfo &
while true
```

```
do
    xsend xinfo setMessage `date`
    sleep 1
done
```

A further example is provided by extending the file browser example of earlier. If a directory is selected then the browser should move to that directory, updating the file contents as it does so. The applet `xselect` responds to the command `setList`, so this gets sent each time a directory change is made:

```
ls | xselect |
while read file
do
    if [ -d $file ]
    then
        cd $file
        xsend xselect setList *
    else
        xedit $file &
    fi
done
```

Handling Selections

A very common communication method between X applications has been the "select" and "paste" mechanism, whereby a selection in one application may be pasted into another. A mechanism which is similar in effect but different in appearance is that of "drag and drop" in which a selected item is dragged to another and dropped on it. While there are only a few commonly used selection mechanisms, there is unfortunately more than one drag and drop system. We only use the Motif mechanism here.

The "send" command can be used for programmatic control of one application by another. The selection and drag and drop mechanisms on the other hand are employed by a user to transfer data from one application to another.

The applet `xdrop` receives the result of a drop or paste and converts it to a string if possible and sends it to stdout. This allows a set of independant applications to be run as asynchronous applications with an indeterminate source of information. For example, to run print managers which can print text or files, fire up these two:

```
xdrop -title "print text" | lpr &

xdrop -title "print file" |
while read file
do
    lpr $file
done
```

Such applications could be left as a row along the bottom of the screen, waiting to receive dropped or pasted text or files and print them.

4. Implementation Issues

Requirements

Most of the tools described here could be largely implemented using any of the toolkits, in any language. The requirements for a filter, say, are

1) An applet must be able to process line buffered input as it is ready.
2) An applet must be able to produce line buffered output.
3) To use the "send" mechanism a suitable form of the "send" library needs to be available.

The ability to use the Tk send command in practice limits this to tcl/Tk, tclMotif or to C–based Xt applications such as Motif or Athena applications with the tclXtSend library.

The pseudo–code for a very simplified form of xselect is

```
Create a MessageBox with Quit button
Create a ScrolledList in the MessageBox
Add a callback to list selection
Add an input handler to stdin
Realize widgets
Enter event loop

Input handler:
    if EOF on stdin, remove handler else
       read a line and add it to the end of the
ScrolledList

List callback:
    get selected items
    write them to stdout
```

Implementation

Several experiments were performed. For example, `xask` was implemented in both tcl/Tk and tclMotif. In tcl/Tk it took 82 lines and in tclMotif it took 75 lines*, with insignificant differences in functionality. The applet `xselect` was implemented in both tclMotif and in C. In tclMotif it took 129 lines, whereas the C version was 264 lines*. The C version was syntactically far more complex and required knowledge of creating tcl commands from within C as well as a knowledge of Motif.

The `wish` interpreter used in tcl/Tk programs was 435k bytes in stripped form, dynamically linked to the X11 and maths library. Statically linked, it took 685k. The `moat` interpreter used by tclMotif was 263k in stripped form, dynamically linked to the Motif, Xt, X11 and maths libraries. Statically linked, it took 2.1 Mbytes. The `xselect` applet, when implemented in C, used 123k dynamically linked, and 1.6 Mbytes statically linked. This was using the Motif 1.2 library – the link profile may be expected to be better under Motif 2.0.

*The source code had all comments and blank lines removed

Tests were also run on startup times. This whole area is confused by the use of dynamically linked libraries, the amount of real memory (to avoid swapping, etc) and probably other factors. From a cold boot system, running only the X server, the window manager and an xterm, on a 486 with 20M RAM under Linux, all of `wish`, `moat` and the C version of `xselect` all took about 5 seconds to start. On a repeated run, probably with the same programs and libraries in RAM, these results were obtained all applets came up with no noticeable delay. So it would appear that initial loading of libraries can cause a significant slowdown, but not if they stay resident.

These applets were built using the Motif toolkit. In general the richness of this toolkit made it easier than it may have been with other toolkits. in particular, the large variety of convenience functions in Motif made it very easy to implement xask, xerror, etc.

The most dificult to implement in tclMotif was `xiselect`, and this was due to often commented problems in the Motif widget set. This particular applet displays a list of icon and text string pairs. The Motif List widget will not do this as it only allows strings as list elements. The Motif Label widget can display a string or a pixmap, but usually not both. In fact, there is a case where it will display both, and that is when the type is set to pixmap but there is also an accelerator set. The Label will show both the pixmap and the text. Unfortunately, this option is only available when a Label is used in a menu so that it is not of general use.

The only possible way of showing both is to use a DrawnButton widget, where the type is set to pixmap and the string is drawn using the Xlib `XDrawImageString` function. The drawback with this is that the widget itself has no idea of the size of the string that will be drawn in it as this is not one of the widget's resources, and so this size cannot be used by the widget in size calculations for the widget. This causes a problem when trying to decide on a suitable container for this list of buttons. Ideally, it would behave like the Motif ScrolledList, with a minimum size of the longest string and a horizontal scrollbar if the width of the ScrolledWindow container becomes less than this. This works with List because it can calculate this minimum size. It does not work with buttons whose size is unknown. This was resolved by linking the size of the buttons to the ScrolledList clip window, so that the user was ultimately responsible.

5. Future Work

Anthony Thyssen (anthony@cit.gu.edu.au) has suggested applications already available such as `xmessage`, `notifier`, `xprompt`, `xmenu` and `xedit` as suitable additions. Presently, these do not support the ''send'' library, but it should be an easy matter to add this support. The addition of these and others could extend the scope of XScript.

The ''send'' mechanism may be used to interactively connect applications. A ''connecting'' applet could be used to draw a line between a pair of applets, and then inform the first that it is to send output to the second. I would like to thank one of the referees for this suggestion.

6. Conclusion

This paper has discussed how small applets can be constructed which allow graphical objects to be used from the shell command line, or within shell scripts. Such applets form an extensible set. This allows shell programmers to create scripts which take advantage of the X Window System without having to learn (or use) X programming.

The system has been used for several years by the author in student assignments for learning shell and C programming, where the tools allow more interesting projects than in a character–based environment. For example, in a project to demonstrate IPC by controlling the running state of an application, a C program set up a pipeline from `xask` and sent SIGSTART and SIGSTOP signals to the application based on whether the user pressed the Yes or No buttons.

7. Availability

The XScript system is available from X ftp sites such as ftp.x.org, and includes standard Unix man pages for each applet. The primary site is ftp://ftp.canberra.edu.au/pub/motif/xscript. The current version is 1.1, and it requires Motif 1.2, tcl 7.3 and tclMotif 1.4. tcl and tclMotif are both available from ftp.x.org.

8. References

[1] D. Blewett, S. Anderson, M. Kilduff, S. Udovic and M.Wish, *X Widget Based Software Tools for UNIX*, Proc Winter USENIX, 1992

[2] K. Buehler, *The xgen Application Generator: Constructing Motif GUIs for TTY Programs*, The X Resource, issue 4, 1992

[3] D. Heller, *The Motif Programming Manual*, O'Reilly and Associates, 1991

[4] G. Neumann and S. Nusser, *Wafe – an X Toolkit Based Frontend for Application Programs in Various Programming Langauges*, Proc Winter USENIX, 1993

[5] J. D. Newmarch, *A Binding of tcl to Motif*, XHibition 94, San Jose, 1994

[6] J. D. Newmarch, *Using tcl to Replay Xt Applications*, AUUG 94, Melbourne, 1994

[7] J. D. Newmarch, *The tclXtSend library*, anonynous ftp from ftp.x.org/contrib/libraries

[8] J. Ousterhout, *Tcl and the Tk Toolkit*, Addison–Wesley, 1994

[9] K. Sall, B. Dealy and J. D. Newmarch,*The Motif Frequently Asked Questions*, anonymous ftp from rtfm.mit.edu:/pub/usenet/comp.windows.x.motif

[10] D. Smyth, *Wcl 2.0: The Widget Creation Library* The X Resource, issue 2, 1992

[11] M. G. Sobell, *A Practical Guide to the Unix System*, Benjamin–Cummings, 1991

[12] L. Virden, *The Tcl Frequently Asked Questions*, anonymous ftp from rtfm.mit.edu:/pub/usenet/comp.lang.tcl

[13] J. S. Pendergrast Jnr, *The CDE Desktop Koorn Shell*, XHibition 94, San Jose, 1994

[14] Byte applets