9
TTY Subwindows

The TTY (or *terminal emulator*) subwindow emulates a standard terminal, the principal difference being that the row and column dimensions of a TTY subwindow can vary from that of a standard terminal. In a TTY subwindow, you can run arbitrary programs, including a complete interactive shell. Or you can emulate terminal interface applications that use the *curses*(3X) terminal screen optimization package without actually running a separate process. The TTY subwindow accepts the standard ANSI escape sequences for doing ASCII screen manipulation, so you can use *termcap* or *termio* screen-handling routines. This chapter discusses the TTYSW package. Figure 9-1 shows the class hierarchy for the TTYSW package.

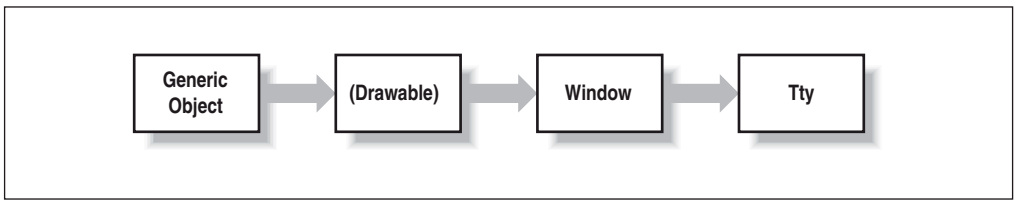

Figure 9-1. TTY package class hierarchy

9.1 Creating a TTY Subwindow

Programs using TTY subwindows must include the file <*xview/tty.h*>. Like all XView windows, you create a TTY subwindow by calling xv_create() with the appropriate type parameter, as in:

```
Tty tty;
tty = xv_create(frame, TTY, NULL);
```

The default TTY subwindow will fork a shell process and the user can use it interactively to enter commands. This program does not interact with the processing of the application in which the TTY subwindow resides; it is an entirely separate process. For example, if you want to start the TTY subwindow with another program, say *man*, you can do so by specifying the name of the program to run via the TTY_ARGV attribute, as shown in Example 9-1.

Example 9-1. The sample_tty.c program

```
/*
 * sample_tty.c -- create a base frame with a tty subwindow.
 * This subwindow runs a UNIX command specified in an argument
 * vector as shown below.  The example does a "man cat".
 */
#include <xview/xview.h>
#include <xview/tty.h>

char *my_argv[ ] = { "man", "cat",  NULL };

main(argc, argv)
char *argv[ ];
{
    Tty tty;
    Frame frame;

    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL);
    frame = (Frame)xv_create(NULL, FRAME, NULL);
    tty = (Tty)xv_create(frame, TTY,
        WIN_ROWS,       24,
        WIN_COLUMNS,    80,
        TTY_ARGV,       my_argv,
        NULL);

    window_fit(frame);
    xv_main_loop(frame);
}
```

The output of Example 9-1 is shown in Figure 9-2. Note that you can have only one TTY subwindow per process.

9.2 Driving a TTY Subwindow

You can drive the terminal emulator programmatically. There are procedures both to send input to the terminal emulator (as if the user had typed it in the TTY subwindow) and to send output (as if a program running in the TTY subwindow had output it). You can send input to a TTY subwindow programmatically with the function:

```
int
ttysw_input(tty, buf, len)
    Tty   tty;
    char *buf;
    int   len;
```

`ttysw_input()` appends the character sequence in `buf` that is `len` characters long onto `tty`'s input queue. It returns the number of characters accepted. The characters are treated as if they were typed from the keyboard in the TTY subwindow. `ttysw_input()` pro-

vides a simple way for a window program to send input to a program running in its TTY subwindow. You can send output to a TTY subwindow programmatically with the function:

```
int
ttysw_output(tty, buf, len)
    Tty   tty;
    char *buf;
    int   len;
```

`ttysw_output()` runs the character sequence in `buf` that is `len` characters long through the terminal emulator of `tty`. It returns the number of characters accepted. The effect is similar to executing this:

echo *`character_sequence`* > /dev/tty*N*

where `tty`*N* is the pseudo-TTY associated with the TTY subwindow. `ttysw_output()` can be used to send ANSI escape sequences to the TTY subwindow.

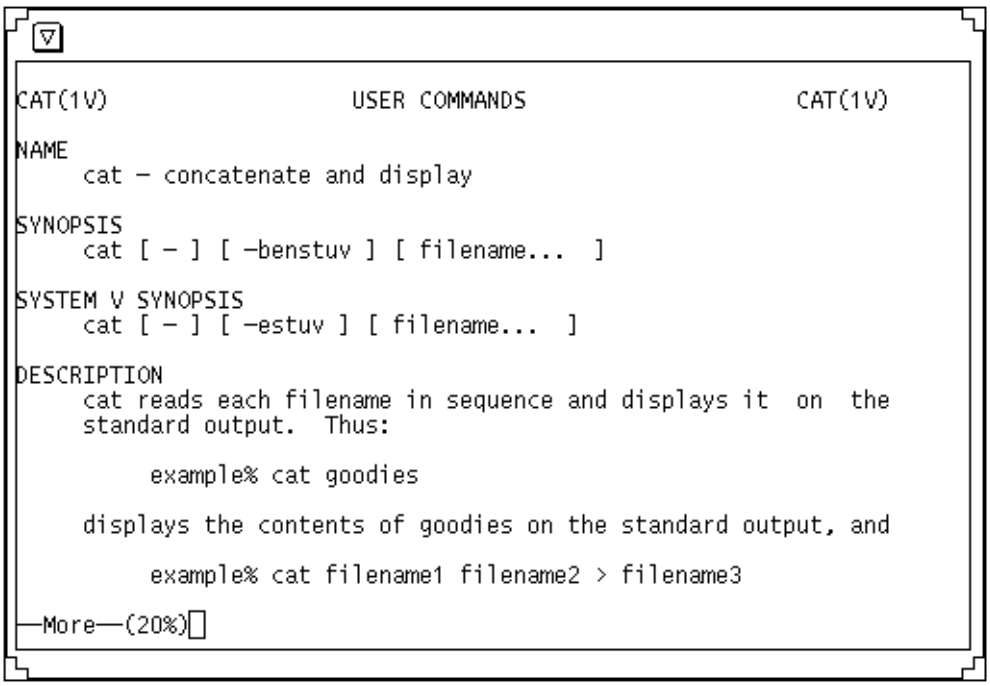

Figure 9-2. Output of sample_tty.c

Note the differences between the input and output TTY routines. If an application is running in the TTY subwindow, then the characters sent to the TTY subwindow using `ttysw_input()` are sent to that program as its `stdin`. Characters sent to the TTY subwindow using `ttysw_ouput()` are sent to the TTY subwindow itself and have nothing to do with the application that might be running in the window.

The program in Example 9-2 creates a text subwindow in which the user can type input. There is a panel button called “Text to TTY” which, if selected, reads the data in the text subwindow and sends it to the TTY subwindow using `ttysw_input()`.

Example 9-2. The textsw_to_ttysw.c program

```
/*
 * textsw_to_ttysw.c -- send text from a text subwindow to a
 * tty subwindow using ttysw_output()
 */
#include <stdio.h>
#include <xview/panel.h>
#include <xview/xview.h>
#include <xview/textsw.h>
#include <xview/tty.h>
Textsw  textsw;
Tty     ttysw;

main(argc,argv)
int     argc;
char    *argv[ ];
{
    Frame       frame;
    Panel       panel;
    void        text_to_tty(), exit();

    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL);

    frame = (Frame)xv_create(XV_NULL, FRAME,
        FRAME_LABEL, argv[0],
        NULL);
    panel = (Panel)xv_create(frame, PANEL,
        PANEL_LAYOUT, PANEL_VERTICAL,
        NULL);
    (void) xv_create(panel, PANEL_BUTTON,
        PANEL_LABEL_STRING,     "Quit",
        PANEL_NOTIFY_PROC,      exit,
        NULL);
    (void) xv_create(panel, PANEL_BUTTON,
        PANEL_LABEL_STRING,     "Text To Tty",
        PANEL_NOTIFY_PROC,      text_to_tty,
        NULL);
    window_fit(panel);

    textsw = (Textsw)xv_create(frame, TEXTSW,
        WIN_ROWS,       10,
        WIN_COLUMNS,    80,
        NULL);
    ttysw = (Tty)xv_create(frame, TTY,
        WIN_BELOW,      textsw,
        WIN_X,          0,
        TTY_ARGV,       TTY_ARGV_DO_NOT_FORK,
        NULL);

    window_fit(frame);
    xv_main_loop(frame);
}
```

Example 9-2. The textsw_to_ttysw.c program (continued)

```
/*
 * callback routine for the panel button -- read text from textsw
 * and send it to the ttysw using ttysw_output()
 */
void
text_to_tty(item, event)
Panel_item item;
Event *event;
{
    char buf[BUFSIZ];

    (void) xv_get(textsw, TEXTSW_CONTENTS, 0, buf, sizeof buf);
    ttysw_output(ttysw, buf, strlen(buf));
}
```

Figure 9-3 shows the output of Example 9-2.

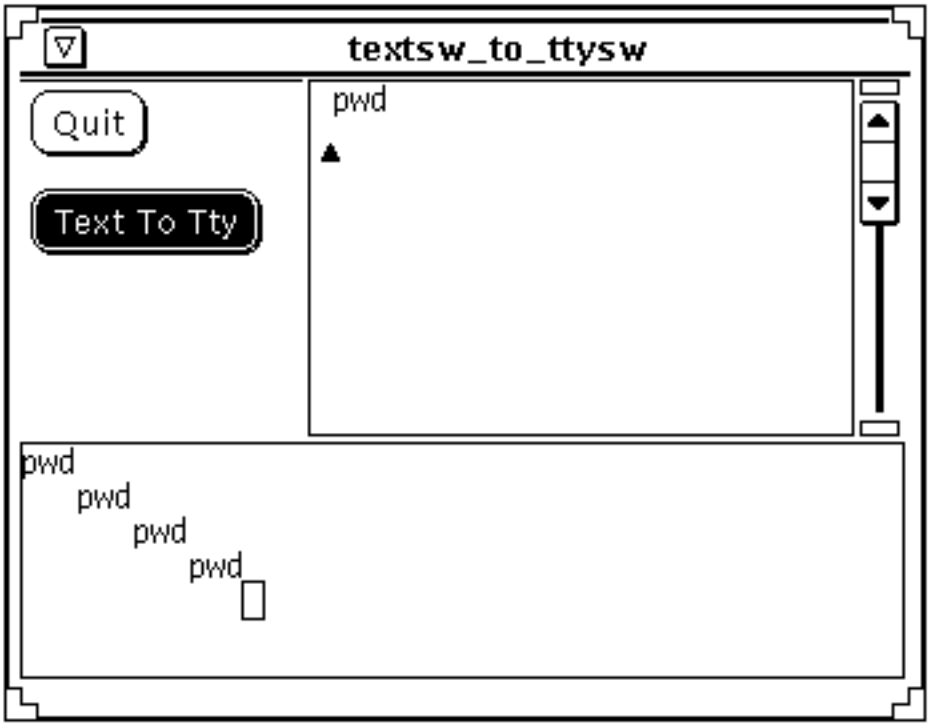

Figure 9-3. Output of textsw_to_ttysw.c

Using `ttysw_output()` shows that the text is simply output to the dummy terminal emulator described by `ttysw`. `ttysw_input()` is useful for sending data as input to a program running in the TTY subwindow. For example, a window-based front end for a text editor could be written where all the common functions such as Save and Next Page can be programmed into panel buttons. Selecting one of those panel buttons would cause a constant string to be sent to the application to be processed as input. The *write filename* function in *vi* could have a button that uses `ttysw_input()` to send the string (w!\n) to the TTY subwindow containing the program.

9.3 Monitoring the Program in the TTY Subwindow

When you use the `TTY_ARGV` attribute to pass the name of a program to run to the TTY subwindow, the program runs as a forked child process. If the attribute:

```
TTY_QUIT_ON_CHILD_DEATH
```

is set to `TRUE`, then the application exits when the forked program exits. But, by default, this attribute is set to `FALSE`. You can use `TTY_PID` to monitor the state of the child process running in the TTY window via the Notifier using `notify_set_wait3_func()`. The client's `wait3()` function gets called when the state of the process in the TTY subwindow changes:*

```
#include <sys/wait.h>
static Notify_value     my_wait3();

    ...
    ttysw = xv_create(base_frame, TTY,
        TTY_ARGV,                   my_argv,
        NULL);
    child_pid = (int)xv_get(ttysw, TTY_PID);
    notify_set_wait3_func(ttysw, my_wait3, child_pid);
    ...
```

The `wait3()` function can then do something useful, such as destroying the TTY window or starting up another process. The code fragment below detects when any of the TTY subwindow's child processes has died.

```
static Notify_value
my_wait3(ttysw, pid, status, rusage)
Tty              ttysw;
int              pid;
union wait      *status;
struct rusage   *rusage;
{
    int    child_pid;

    notify_next_wait3_func(ttysw, pid, status, rusage);
    if (!(WIFSTOPPED(*status))) {
     /* rerun the program */
        xv_set(ttysw, TTY_ARGV, my_argv, NULL);
```

*This includes when the program *stops* in addition to when it exits.

```
            child_pid = (int)xv_get(ttysw, TTY_PID);
            notify_set_wait3_func(ttysw, my_wait3, child_pid);
        }
        return NOTIFY_DONE;
    }
```

You can set `TTY_PID` as well as get it, but if you set it, you are responsible for setting:

```
notify_set_wait3_func()
```

to catch the child's death. You are also responsible for directing the standard input and standard output of the child to the pseudo-TTY.

9.4 Talking Directly to the TTY Subwindow

Setting `TTY_ARGV` to `TTY_ARGV_DO_NOT_FORK` tells the system not to fork a child in the TTY subwindow. In combination with `TTY_TTY_FD`, this allows the tool to use standard I/O routines to read and write to the TTY subwindow by getting the file descriptor of the pseudo-TTY associated with the TTY subwindow. You can then use this to read and write to the pseudo-TTY using standard UNIX I/O routines.

Example 9-3 uses a TTY subwindow to create a pseudo *terminal* in which *curses* routines can be used. Five panel items are displayed. Along with the usual Quit button to exit the program, a Print button displays the text in the text panel item at the coordinates input in the X and Y numeric text items.

Example 9-3. The ttycurses.c program

```
/*
 * ttycurses.c -- An application that uses a tty subwindow that
 * emulates a tty so well, you can use curses(3x) routines in it.
 * This program does not handle resizes -- resizing the base frame
 * produces unpredictable results.  To handle resizing properly,
 * the application should install a resize event handler and
 * call endwin() followed by initscr() to reinitialize curses
 * to reflect the size of the window.
 *
 * cc ttycurses.c -lxview -lcurses -ltermlib
 */
#include <curses.h>
#undef WINDOW /* defined by curses.h -- needs to be undefined */
#include <xview/xview.h>
#include <xview/panel.h>
#include <xview/textsw.h>
#include <xview/tty.h>

/* panel items contain the x,y info for outputting text to the ttysw */
Panel_item  x, y, text;

main(argc,argv)
int     argc;
char    *argv[ ];
{
    Frame       frame;
```

Example 9-3. The ttycurses.c program (continued)

```
    Panel       panel;
    Tty         ttysw;
    char        buf[16];
    void        output(), exit();

    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL);

    frame = xv_create(XV_NULL, FRAME,
        FRAME_LABEL,            argv[0],
        FRAME_SHOW_FOOTER,      TRUE,
        NULL);

    panel = (Frame)xv_create(frame, PANEL, NULL);
    (void) xv_create(panel, PANEL_BUTTON,
        PANEL_LABEL_STRING,             "Quit",
        PANEL_NOTIFY_PROC,              exit,
        NULL);
    (void) xv_create(panel, PANEL_BUTTON,
        PANEL_LABEL_STRING,             "Print",
        PANEL_NOTIFY_PROC,              output,
        NULL);
    x = (Panel_item)xv_create(panel, PANEL_NUMERIC_TEXT,
        PANEL_LABEL_STRING,             "X:",
        PANEL_VALUE_DISPLAY_LENGTH,     3,
        NULL);
    y = (Panel_item)xv_create(panel, PANEL_NUMERIC_TEXT,
        PANEL_LABEL_STRING,             "Y:",
        PANEL_VALUE_DISPLAY_LENGTH,     3,
        NULL);
    text = (Panel_item)xv_create(panel, PANEL_TEXT,
        PANEL_LABEL_STRING,             "Text:",
        PANEL_VALUE_DISPLAY_LENGTH,     10,
        PANEL_VALUE,                    "X",
        NULL);
    window_fit(panel);

    ttysw = (Tty)xv_create(frame, TTY,
        WIN_BELOW,      panel,
        WIN_X,          0,
        TTY_ARGV,       TTY_ARGV_DO_NOT_FORK,
        NULL);
    window_fit(frame);

    dup2((int)xv_get(ttysw, TTY_TTY_FD), 0); /* dup2 closes 0 first */
    dup2((int)xv_get(ttysw, TTY_TTY_FD), 1); /* dup2 closes 1 first */

    /* initscr() initializes the curses package and determines
     * characteristics about the window as if it were a terminal.
     * The curses specific variables, LINES and COLS are now set
     * to the row and column sizes of the window.
     */
    initscr();

    xv_set(x, PANEL_MAX_VALUE, COLS-1, NULL);
    xv_set(y, PANEL_MAX_VALUE, LINES-1, NULL);
    sprintf(buf, "LINES: %d", LINES-1);
```

Example 9-3. The ttycurses.c program (continued)

```
    xv_set(frame, FRAME_LEFT_FOOTER, buf, NULL);
    sprintf(buf, "COLS: %d", COLS-1);
    xv_set(frame, FRAME_RIGHT_FOOTER, buf, NULL);

    xv_main_loop(frame);
}
/*
 * callback routine for the <print> panel button.  Get the coordinates
 * and the text to print on the tty subwindow and use curses library
 * routines to render the text.
 */
void
output()
{
    int X = (int)xv_get(x, PANEL_VALUE);
    int Y = (int)xv_get(y, PANEL_VALUE);
    char *Text = (char *)xv_get(text, PANEL_VALUE);
    mvaddstr(Y, X, Text);
    refresh();
}
```

9.5 TTY Subwindow Function Key Escape Sequences

XView provides a default *.ttyswrc* file for the `shelltool` which provides the SunView escape key sequences for the following keys: L3, F1-F12, R1-R7, R9, R11, R13, and R15 (for more details see *$OPENWINHOME/lib/.ttyswrc*). To override the default file, place any `.ttyswrc` file in your *$HOME* directory. To avoid using any *.ttyswrc* file, place the following line in your *.Xdefaults* file:

```
term.useAlternateTtyswrc:  False
```

In addition, you will need to remove any *.ttyswrc* files from your *$HOME* directory. To specify a different alternate file than the one in *$OPENWINHOME/lib/.ttyswrc*, place the following into your *.Xdefaults* file:

```
term.alternateTtyswrc:  filename
```

where *filename* is the path and name of the file to use. For example:

```
term.alternateTtyswrc:  /usr/lib/ttyswrc
```

When using a Sun Type 4 keyboard with an X11 server other than OpenWindows 3.0, escape key sequences cannot be produced for keys F11 and F12 due to a limitation in the standard X11 keysym definition file. The base keysym file distributed by the X Consortium limits the number of unique function keys to a maximum of 35 (for more details see *$OPENWINHOME/include/X11/keysym.h*). The Sun Type 4 keyboard has 37 keysyms. For OpenWindows, there are two extra keysyms Sun_F36 and Sun_F37 which allow XView and other X-based programs to perform unique actions on the F11 and F12 keys (for more details see *$OPENWINHOME/include/X11/Sunkeysym.h*).

9.6 TTY Package Summary

The TTYSW procedures are shown in Table 9-1. Table 9-2 lists the TTYSW attributes. This information is described fully in the *XView Reference Manual*.

Table 9-1. TTY Subwindow Procedures

`ttysw_input()`
`ttysw_output()`

Table 9-2. TTY Subwindow Attributes

`TTY_ARGV`
`TTY_ARGV_DO_NOT_FORK`
`TTY_CONSOLE`
`TTY_PAGE_MODE`
`TTY_PID`
`TTY_QUIT_ON_CHILD_DEATH`
`TTY_TTY_FD`
`WIN_FONT`
`WIN_SET_FOCUS`