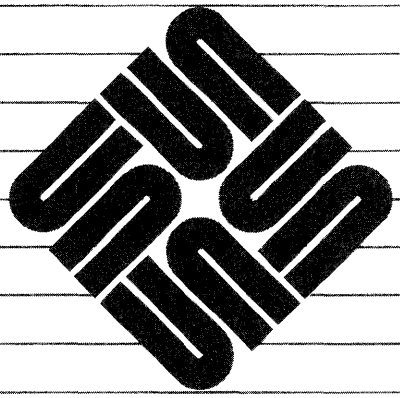
sun microsystems

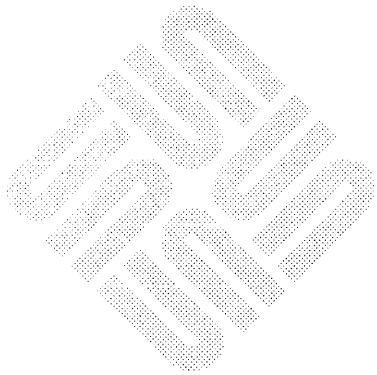
SunView 1.80 Update Appendix



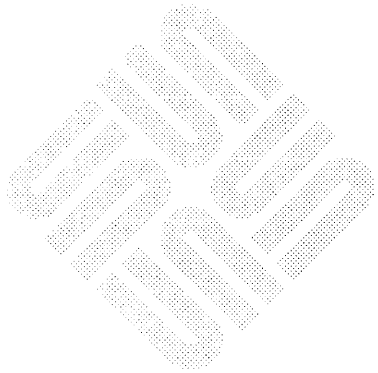
Part Number: 800-4738-10
Revision A of 27 March, 1990

Contents

Figures



Tables



D

SunView 1.80 Update

The major features of SunView 1.80 are described in detail here:

- an **online help mechanism**, allowing application developers to provide Spot Help for their users,
- **programmable alarms** for dramatically notifying users,
- keyboard support
  - type 4 keyboard
  - upgraded description of the `.textswrc` file
- enhanced color capabilities
  - **colored panel items**,
  - support for **24-bit true color**,
- changes to the defaults database
- several user changes,
- various bug fixes.

Not described in this Update are changes to the Defaults Editor database or new user features, which are contained in the *SunView User's Guide*.

Note that there is not a separate update document for the *SunView System Programmer's Guide*, the information for which appears here.

D.1. SunView Help Mechanism

The new release of SunView offers two related mechanisms for providing online help to users:

- Spot Help, a cursor-position sensitive facility to display one 32 × 80 character panel of online help,
- More Help, to provide additional information when the one panel of Spot help is not enough.

Basics of Spot Help in SunView 1.80

To get help, the user places the pointer over the object (panel, button, etc) of inquiry and then strikes the Help key. Whatever information is available is then displayed.

Help Keys

On a Type3 keyboard, the Help Key is Meta-/, obtained by pressing the Meta key and the / key at the same time. There are two Meta keys, which are immediately to the left and right of the long space bar on the bottom center of the keyboard.

On a type 4 keyboard, the Meta-/ combination works, and there is an explicit Help key also.

- The Meta keys are in the same place as on the Type 3 keyboard, beside the space bar, marked with a diamond, ♦.
- Help is the double-width key located at the bottom of the left hand block of function keys (the ones labeled Stop, Again, etc).

Limitations of Spot Help

There are two limitations to the use of Spot Help on SunView 1.80:

- At this time, **only the mechanism** for Spot Help is provided; *no actual Help Text is provided.* Available resources do not allow the development of Help Text for SunView itself, but the mechanism is being made available to developers who want to provide cursor-position sensitive help in their applications.
- Also note that Spot Help supports a single window of text, 32 lines by about 80 characters (longer lines are not supported at this time). To obtain longer messages, you must use the More Help feature. This is a user implemented feature called by the **More** button on the help window. See *More Help*, below.

The Help Directory

A new category of defaults, `Help`, has been added to `Default Editor` to support Spot Help. Inside this category, the default `Help/Directory` is used to identify the directory where Help Text for an application resides. By default, this directory is `/usr/lib/help`, but it can be changed to any directory.

Help Text: A Simple Example

The following simple experiment will show you how to add Help Text for a SunView text subwindow. This experiment is intended only as a quick way to see the action of Spot Help.

Bring up a tool that uses a text subwindow (`textedit`, or `mailtool`, for example). Place the cursor in the subwindow and press Help. You should see a message saying:

```
No help is available for textsw:textsw.
```

To remedy this, create a file named `textsw.info` in the `/usr/lib/help` directory. (You can create the file anywhere else if you remember to change the Defaults Editor `Help/Directory` entry to point to it.) Put in the key `:textsw` and the Help Text you want. For example:

```
:textsw:
1. This is a text subwindow
 line 2
 line 3
4. line 4
```

Now bring up a new tool containing a text subwindow, again position the cursor in the window, and press [Help].

This time you should see the text you entered in the file.

You can do this for any feature in SunView: put the cursor over the item and press [Help] to see the message:

```
No help is available for package:feature .
```

In the example, the filename and the key were both `textsw`. In general, of course, this is not the case.

Create (or append to) a file named *package* `.info` an entry following the keyword *:feature*.

The `.info` file has the following format:

 # comments
:keyword1 [*keyword2* [*keyword3*]] [*:more_help_key*]
message text

You can include comment lines in your `.info` files by preceding them with the number sign. Use an initial colon to denote a line containing a keyword or keywords. If several keywords pertain to the same help message, place them on the same line, with spaces separating them. The message text supplied appears in the Spot Help window whenever this `.info` file and *keyword1*, *keyword2*, or *keyword3* are values for the `HELP_DATA` attribute.

Several examples of `.info` files are shown below, following the discussion of the Spot Help mechanism.

Spot Help Program Interface

This section explains how to create Spot Help messages for text subwindow, panel, canvas, alert, tty, and menu window objects, as well as for individual menu, scroll bar, and panel items. It assumes you are familiar with SunView programming concepts; for more information, consult the *SunView Programmer's Guide*

The two basic steps to include Spot Help for a window object are:

1. Add the `HELP_DATA` attribute to the object or to an item within the object. You can add this attribute like other SunView attributes, such as through a null-terminated attribute list
2. Write the help file in the format specified above.

When a user presses the key, the HELP_DATA attribute is retrieved from the current window or item. The text specified by the HELP_DATA value is then displayed in the Spot Help window.

The value for the HELP_DATA attribute must be a two-part string, enclosed in quotation marks, in the format:

```
"file:keyword"
```

file is the name of the text file containing the help description. *file* must be located in the default help directory and must end with the suffix `.info` (such as `myapplication.info`). Although all Spot Help files must end with the `.info` extension, include only the base of the file name, **not the extension**, as the value of the HELP_DATA attribute. The Help mechanism automatically appends" the `.info` extension to the file name that you supply, and then looks in the default help directory (`/usr/lib/help` initially) for that file.

`keyword` is a word within the `.info` file that is associated with the specific help text that will appear when help is requested. Each `.info` file can contain multiple keywords, but no two keywords can be alike within the same `.info` file.

For example, a HELP_DATA attribute could be

```
HELP_DATA, "accounting:w4"
```

When help is requested on this object, the Help facility:

1. Finds the `accounting.info` file.
2. Locates the keyword `w4`.
3. Displays the text associated with that keyword in a Spot Help window.

The `.info` File Format section contains more details about the structure and placement of `.info` file text. The next section describes how you can use the HELP_DATA attribute to make your Spot Help messages more helpful for users.

Providing More Specific Spot Help

You can change the HELP_DATA attribute of various window objects to suit particular circumstances, for instance if a menu item is active or disabled, or a frame is open or iconic. If you do, you can provide users with more context-sensitive Spot Help, as described in this section.

HELP_DATA for Active and Disabled Objects

For example, you might give all disabled objects (such as greyed-out menu items) a new HELP_DATA attribute where you disable them in the code, and again where you activate them, as described below:

```
/* this menu item invokes a save function */
Menu_item       mi_save;
      . . .
/* here the save function becomes disabled */
menu_set (mi_save, MENU_INACTIVE, TRUE,

HELP_DATA, "myapp:mi_save_disabled",
            0);
      . . .
/* and here it becomes active */
menu_set (mi_save, MENU_INACTIVE, FALSE,

HELP_DATA, "myapp:mi_save",
            0);
```

A corresponding Spot Help message for the "save" function above could be:

```
Save menu item
Stores the current version of the file you have loaded.
```

The Spot Help message when the save function is disabled could be:

```
Save menu item [DISABLED]
Stores the current version of the file you have loaded.
This item is disabled because you have not loaded a file.
```

The `myapp.info` file to display the above messages would look like

```
:mi_save
Save menu item
Stores the current version of the file you have loaded.
:mi_save_disabled
Save menu item [DISABLED]
Stores the current version of the file you have loaded.
This item is disabled because you have not loaded a file.
```

Alternatively, a single message might be used to cover Spot Help for both situations. This is achieved by a multiple-key entry in the `myapp.info` file, such as:

```
:mi_save mi_save_disabled
Save menu item
Stores the current version of the file you have loaded.
This item is disabled if you have not loaded a file.
```

You also could include HELP_DATA attributes for frames that are open and those that are icons (closed). The following sample program creates a base frame and then interposes an event function in front of the frame's normal event handler. This makes the program aware of when the frame opens or closes, as well as when the program should change the frame's HELP_DATA attribute.

```
#include <suntool/sunview.h>
#include <suntool/help.h>
#include <stdio.h>
main(argc, argv)
    int             argc;
    char            **argv;
{
    Frame           frame;
    Notify_value    sample_interpose();
/* create frame using command-line arguments */
   frame = window_create(0, FRAME, FRAME_ARGS,
     argc, argv, 0);
/* set HELP_DATA depending on whether frame is
      open or iconic */
   if ((int)window_get(frame, FRAME_CLOSED)) {
      window_set(frame,

HELP_DATA, "progname:frame_iconic" ,
    0);
   } else {
      window_set(frame,

HELP_DATA, "progname:frame" ,
    0);
   }
/* interpose in order to spot future open/close events */
   (void)notify_interpose_event_func(frame,
     sample_interpose, NOTIFY_SAFE);
   window_main_loop(frame);
}
static Notify_value
sample_interpose(frame, event, arg, type)
    Frame           frame;
    Event           *event;
    Notify_arg      arg;
    Notify_event_type type;
{
    int             initial_state, current_state;
    Notify_value    value;
/* get frame's state */
   initial_state = (int)window_get(frame,FRAME_CLOSED);

/* handle the event */
   value = notify_next_event_func(frame, event,arg, type);

/* if frame's state has changed, change HELP_DATA */
   current_state = (int)window_get(frame,FRAME_CLOSED);
```

```
   if (initial_state != current_state) {
      if (current_state) {  window_set(frame,

HELP_DATA,"progname:frame_iconic" ,
    0);
      } else { window_set(frame,

HELP_DATA,"progname:frame" ,
    0);
        }
    }
    return(value);
}
```

Spot Help Example

The following program puts up a SunView window with several panel items and buttons and Spot Help for them.

The beginning of the main loop includes some header files and defines some storage and some SunView objects.

```
/* client.c
 * Constructs a simple panel, showing use of
 * HELP_DATA attributes.
 */
#include <stdio.h>
#include <suntool/sunview.h>
#include <suntool/panel.h>
#include <suntool/help.h>

main(argc, argv)
    int     argc;
    char        **argv;
{
    Frame       frame;
    Panel       panel;
```

Note the use of the HELP_DATA attribute here. This is where the link to the Help Text in the file is actually made.

```
frame = window_create(NULL, FRAME,
  FRAME_LABEL, argv[0],
  FRAME_ARGS, argc, argv,

"HELP_DATA, "client:frame" ,
  0);
panel = window_create(frame, PANEL,
  WIN_WIDTH, 200,
  WIN_HEIGHT, 200,

"HELP_DATA, "client:panel" ,
  0);
panel_create_item(panel, PANEL_TEXT,
  PANEL_LABEL_STRING, "Year:",
  PANEL_VALUE, "1988",

HELP_DATA, "client:year" ,
  0);
panel_create_item(panel, PANEL_TEXT,
  PANEL_LABEL_STRING, "Maker:",
  PANEL_VALUE, "Ford",

HELP_DATA, "client:maker" ,
  0);
panel_create_item(panel, PANEL_TEXT,
  PANEL_LABEL_STRING, "Model:",
  PANEL_VALUE, "Escort",

HELP_DATA, "client:model" ,
  0);
```

Note also that there are no callback procedures defined for the buttons. In a more real-life example, of course they would be used.

```
panel_create_item(panel, PANEL_BUTTON,
  PANEL_LABEL_IMAGE,
  panel_button_image(panel, "Find", 0, 0),

HELP_DATA, "client:find_button",
  PANEL_ITEM_X, 40,
  PANEL_ITEM_Y, 160,
  0);
panel_create_item(panel, PANEL_BUTTON,
  PANEL_LABEL_IMAGE,
  panel_button_image(panel, "Done", 0, 0),

HELP_DATA,
 PANEL_ITEM_X, 110,
 PANEL_ITEM_Y, 160,
 0);
window_fit(frame);
window_main_loop(frame);
}
```

The following is the `client.info` file containing the Help Text for the `client` "application". The point is to notice how the keys in this file are delimited (:) and how they connect the text in this file to the objects in `client.c` marked with the `HELP_DATA` attribute.

Most of the keys in this example also have a second colon (:) and a second string associated with them. This string is used to invoke *More Help*, the second feature of the SunView 1.80 help mechanism.

Note the `:find_button` keyword. It has no text, but does have a second colon and string following it. This is a shortcut to More Help.

You have to include the blank lines if you want spacing in the Spot Help message.

```
:frame
:More_About_the_Frame

Sample Help Client

This is the client's frame.

:panel
:More_About_the_Panel

Sample Help Client

This is the client's panel.

:year:
More_About_the_Year_Field

Sample Help Client

This is the client's 'Year' field.


:maker:

Sample Help Client

This is the client's 'Maker' field.
(Notice that 'More Help' is not provided
for this item.)

:model
:More_About_the_Model_Field

Sample Help Client

This is the client's 'Model' field.

:find_button:** Direct help on Find button. **
:done_button
:More_About_the_Done_Button

Sample Help Client

This is the client's 'Done' button.


:end_of_file
```

Here is a key with no Help Text, but instead a More Help string.

More Help

More Help is used when a single panel (32 lines × 80 characters) of Help Text does not suffice. It also allows you to provide a hypertext help facility, **if you choose to write it.**

When More Help is provided, the Spot Help panel comes up with a button saying "More Help". The user who wants more help clicks the mouse over this button, and SunView either finds or tries to start a *More Help server*. Specifically, SunView tries to establish an RPC socket link to the More Help server and to pass to it the More Help string found after the second colon in the `.info` file.

A general More Help server is not provided with SunView 1.80. Programmers needing to include More Help in their application(s) must write their own.

An example is given below that shows how to hook up a More Help server to the SunView mechanism.

Once the server is written,

1. the server executable must be placed in a directory where it can be found in the user's search path, and
2. the server's name must be registered in the `/Help/Server` default in the Defaults Editor database.

This allows SunView to find and start the server when a user asks for More Help and the server is not running.

When SunView starts the server, it uses the equivalent of a command like:

```
server_default_name   More_Help_string
```

where

- *server_default_name* is the name of the server, and
- *More_Help_string* is the More Help information. SunView would have sent this string to the server via RPC but could not because the server was not running. So SunView starts the server and sends the string as a command line argument.

As an example, assume the user is running the `client` program discussed in the previous section, with the Help file `client.info`. Also assume a More Help server named `my_server`.

```
my_server   More_About_the_Panel
```

This command line would be produced in the following way:

1. The user requests Spot Help on the panel.
2. SunView looks in the `.info` file, finds the following entry, which has a More Help string:

```
":panel:More_About_the_Panel"
```

3. So, a **More Help** button is displayed in the Spot Help panel.
4. The user clicks over the **More Help** button.
5. SunView tries to send the More Help string `More_About_the_Panel` to the More Help server (`my_server`).
6. The server is not running, so SunView starts it with the command line shown.

You can pass command line arguments to the server by making them part of the *server_default_name*. For example, having as the `/Help/Server` default

```
my_server -flag1
```

would invoke `my_server` with `flag1` as a command line argument, adding the appropriate More Help string found in the `.info` file when the user requested More Help. In this case, a user wanting More Help about the panel, might result in the More Help server being started with a command line like:

```
my_server   -flag1   More_About_the_Panel
```

Once the server is running, it will display More Help for all SunView applications on demand.

More Help Functions

Three SunView functions support More Help. The first is:

`help_rpc_register` for a More Help function

```
int
help_rpc_register(
more_help_func
)
      void  (*
more_help_func
)();
```

This registers your server with the help system and causes *more_help_func* to be called whenever a help request is generated by the help system. *more_help_func* should be of the form:

```
void

more_help_func
(request_string)
      char  *
request_string
;
{
}
```

where *request_string* is a null terminated character string. The character variable *request_string* will contain the value *more_help_string* that was found with the

Spot Help key `.info` file:

```
    :spot_help_key:
more_help_string
    text
    . . .
    text
```

This function interprets *more_help_string* according to the needs of the application. It could use the string as a key for lookup in a file, along the lines of Spot Help. Or the string could be interpreted as both a filename and a key. Or, the string could be used as a record ID in a database query.

The second SunView More Help function is:

`help_set_more_func` registers the function that gives help on More Help

```
    void
    help_set_more_func(help_on_help_func)

      void (* help_on_help_func)();
```

where *help_on_help_func* is a function similar to the one registered with `help_rpc_register()`, above. It is called when the user asks for help on the More Help server itself. In the simplest case, the function that handles Spot Help requests on the more Help server can be the same one that was registered with `help_rpc_register()`.

The last support function unregisters the More Help server. It is good practice to use this call when the More Help server completes, to release any RPC sockets it used.

`help_rpc_unregister` deregisters the help function

```
    help_rpc_unregister(func);
```

This function should be called before exiting.

More Help Example

The following program shows how to connect a More Help server to the Spot Help mechanism via RPC. When this is done properly, the program will receive the More_Help_string from the `.info` file.

Use of the More Help string is a decision for the application writer.

What this program does with the More Help string is not particularly exciting; it simply makes a panel and displays the string. In more real-life situations, the string might be used as a keyword into a file, or as a filename-keyword pair, or it might be a record ID for a database query.

```
/* server.c */
#include <stdio.h>
#include <suntool/sunview.h>
#include <suntool/panel.h>
#include <suntool/help.h>
Panel_item      What_I_Got;
main(argc, argv)
    int         argc;
    char        **argv;
{
    Frame       frame;
    Panel       panel;
    void        server_rpc_in();
    void        server_local_req();
    frame = window_create(NULL, FRAME,
        FRAME_LABEL, argv[0],
            FRAME_ARGS, argc, argv,
            0);
    panel = window_create(frame, PANEL,
            WIN_WIDTH, 400,
            WIN_HEIGHT, 100,
            0);
    What_I_Got = panel_create_item(panel, PANEL_TEXT,
            PANEL_LABEL_STRING, "What I Got Was: ",
            PANEL_VALUE, "",
            0);
    window_fit(frame);

"help_rpc_register(server_rpc_in)";
    window_main_loop(frame);

"help_rpc_unregister(server_rpc_in)";

}
/*  RPC handler */
static void
server_rpc_in(request_string)
    char            *request_string;
{
    panel_set(What_I_Got, PANEL_VALUE, request_string, 0);
    return;
}
```

Note the use of the function calls.

The help routine registered above, is defined here. It simply creates a panel and displays the More Help string.

Help on the More Help Server

The following code extends the previous example by showing how to provide Spot Help on a More Help server itself. This is done by adding:

- HELP_DATA values for the items needing Spot Help,
- defining a function to handle Spot Help,

- registering the function with `help_set_more_func ()`.

Notice that in this case the Spot Help function simply turns around and calls the same user function that displays More Help.

```
/* server.c */
#include <stdio.h>
#include <suntool/sunview.h>
#include <suntool/panel.h>
#include <suntool/help.h>
Panel_item      What_I_Got;
main(argc, argv)
    int         argc;
    char        **argv;
{
    Frame       frame;
    Panel       panel;
    void        server_rpc_in();
    void        server_local_req();
    frame = window_create(NULL, FRAME,
        FRAME_LABEL, argv[0],
        FRAME_ARGS, argc, argv,

HELP_DATA, "server:frame" ,
            0);
    panel = window_create(frame, PANEL,
            WIN_WIDTH, 400,
            WIN_HEIGHT, 100,

HELP_DATA, "server:panel" ,
            0);
    What_I_Got = panel_create_item(panel, PANEL_TEXT,
            PANEL_LABEL_STRING, "What_I_Got_Was: ",
            PANEL_VALUE, "",

HELP_DATA, "server:What_I_Got" ,
            0);
    window_fit(frame);
    help_rpc_register(server_rpc_in);

help_set_more_func(server_local_request);

    window_main_loop(frame);
    help_rpc_unregister(server_rpc_in);

}
```

Note the addition of `HELP_DATA`; the Help Text is in a file named `server.info`.

Use `help_set_more_func` to define the function to handle Spot Help requests on this More Help program.

The help routines registered above, are defined here.

```
/*  RPC handler */
static void
server_rpc_in(request_string)
    char            *request_string;
{
    panel_set(What_I_Got, PANEL_VALUE, request_string, 0);
    return;
}
/*  required when user asks for help on server itself */
static void
server_local_request(window, request_string, event)
    Window          window;
    char            *request_string;
    Event           *event;
{
    server_rpc_in(request_string);
    return;
}
```

D.2. Programmable Alarms

SunView 1.80 provides programmable alarms, which "beep" and "flash" at the user in a way that is settable from either a C program or from shell commands.

CAUTION

SunView must be installed *and be running* for the alarms to occur, even though you can manipulate the environment variable without SunView.

A beep is the sounding of the bell on the user's keyboard. A flash is a color reversal in a window; the window frame is repainted with the colors reversed, and then painted again normally.

- The number of beeps and the number of flashes can be independently set.
- There is one setting, however, for the duration of both beeps and flashes, and that setting is also the interval between successive beeps/flashes.

Note that the `defaultsedit(1)` values for `SunView/AudibleBell` and `SunView/VisibleBell` will determine whether beeps and flashes, respectively, occur at all. When an aspect of the alarm is disabled by the indicated default, that aspect will not occur, no matter what the setting of the alarm.

Shell Command Interface

SunOS 4.1 provides shell commands to set and get the characteristics of the alarm, and to ring it. These commands rely on an environment variable:

Don't forget the : (colon) characters if you try to enter the setting by hand. You need them at the beginning, end, and in the middle.

```
% set WINDOW_ALARM=:beeps=b:flashes=f:dur=t:
#   where
#   b    = number of beeps
#   f    = number of flashes
#   t    = duration of each beep/flash in milliseconds
```

The setting of this variable can be performed either directly, or through the command:

set_alarm: [**-b** *b* **-f** *f* **-d** *t*]

where the option arguments correspond to the fields in the environment variable.

There is a counterpart command that returns the setting, in the form shown above:

get_alarm

And there is a command to actually ring the alarm:

ring_alarm

This command gets the attributes from `WINDOW_ALARM` and rings the alarm with these attributes. The alarm's behavior is controlled by the SunView `defaultsedit(1)` entries `SunView/Audible_Bell` and `SunView/Visible_Bell`, so the sound and flash can be disabled by the user, regardless of `WINDOW_ALARM`.

`set_alarm` parses its arguments, encodes them into a `termcap(3X)`-like string, and gives to standard output commands to set the environment. The output depends on the value of the `SHELL` environment variable.

```
#For the C shell:
        set noglob;
        setenv WINDOW_ALARM 'string';
        unset noglob;

#For the Bourne shell:
        export WINDOW_ALARM;
        WINDOW_ALARM='string';
```

As a result of the above, the `set_alarm` command must be used in a different manner than other commands (analogous to `tset(1)`). For the Bourne shell and C shells, use this command to place the result of the call to `set_alarm` into the environment for future reference by the library:

```
eval 'set_alarm [options...]'
```

With the C-Shell, it may be convenient to make an alias of the form:

```
alias alarm 'eval 'set_alarm !*''
```

Program Interface

The interface to SunView programmable alarms consists of two calls using the `WIN_ALARM` attribute with the appropriate data structure.

Data Structure

Shown below is the definition of a simple structure type, `Win_alarm` (which is in `window.h`).

```
typedef struct alarmval {
        int     beep_num;
        int     flash_num;
        struct timeval beep_duration;
} Win_alarm;
```

The values of the structure entries are:

- `beep_num` is how many times to beep,
- `flash_num` is how many times, to flash.
- `beep_duration` is how long each individual beep/flash lasts. This is also the elapsed interval between each successive beep and/or flash.

Function Calls

The following call parses the environment variable `WINDOW_ALARM` and returns a pointer to the `Win_alarm` structure.

```
alarm = (Win_alarm *) window_get(window, WIN_ALARM);
```

If `WINDOW_ALARM` is not set, it returns in the `Win_alarm` structure the default values of:

```
beep_num = 1;
flash_num = 1;
beep_duration.tv_sec = 1;
beep_duration.tv_usec = 0;
```

If any of the fields in `WINDOW_ALARM` has an illegal value, `window_get` returns the default value for that field in `Win_alarm`.

If the duration number is not set and either beep or flash is greater than zero, then a default duration of 1 second will be returned in the structure.

The following beeps the keyboard bell and flashes the window frame.

```
window_set(window,WIN_ALARM,&alarm,0)
```

If `&alarm` is NULL, then SunView looks at the environment variable `WINDOW_ALARM` and uses those values to ring the alarm. Again, if the `WINDOW_ALARM` environment variable is not set, SunView will use the default values.

Thus, `window_set(window, WIN_ALARM, 0, 0)` is essentially ringing the alarm with the values from the environment variable; it can beep, flash or both.

The alarm's behavior is controlled by the SunView `defaults_edit(1)` entries `SunView/Audible_Bell` and `SunView/Visible_Bell`, so the

sound and flash can be disabled by the user, regardless of what the call to `win_alarm` specifies.

Programmable Alarm Example

The following example brings up a SunView window with three text items and a button. The text items allow you to specify the characteristics of an alarm, and the button allows you to activate it.

This section defines the default attributes of the alarm, includes the relevant `.h` files, and defines several data structures.

```
#include    <suntool/sunview.h>
#include    <suntool/panel.h>
#define     BEEP_ITEM       0
#define     FLASHES_ITEM        1
#define     DURATION_ITEM       2
#define     NUMBER_OF_ITEMS     3

void    ring_the_alarm();

Frame           frame;
Panel           panel;
Panel_item      panel_items[NUMBER_OF_ITEMS];
Win_alarm       example_alarm;
Pixrect *       rr_button_image;
```

This begins the main loop, which will create a frame and a panel, and define several panel text items.

```
int
main()
{
  frame = (Frame) window_create( (Frame) NULL, FRAME,
    FRAME_LABEL,    "Programmable Alarms Example",
    0  );
  if (frame == (Frame) NULL ){
    fprintf(stderr, "SunView not available\n");
    exit(1);
  }
  panel = (Panel) window_create( frame, PANEL,
    0  );
```

Define the panel text item that accepts user input for how many beeps.

```
  panel_items[BEEP_ITEM] =
    (Panel_item) panel_create_item( panel, PANEL_TEXT,
    PANEL_LABEL_STRING,     "beeps per alarm  :",
    PANEL_VALUE_DISPLAY_LENGTH, 10,
    PANEL_VALUE_STORED_LENGTH,  10,
    PANEL_ITEM_X,           10,
    PANEL_ITEM_Y,           10,
    0  );
  /* check for null pointer */
```

Define the panel text item that

accepts user input for how many flashes.

```
panel_items[FLASHES_ITEM] =
  (Panel_item) panel_create_item( panel, PANEL_TEXT,
  PANEL_LABEL_STRING,      "flashes per alarm:",
  PANEL_VALUE_DISPLAY_LENGTH, 10,
  PANEL_VALUE_STORED_LENGTH,  10,
  PANEL_ITEM_X,            10,
  PANEL_ITEM_Y,            35,
  0  );
/* check for null pointer */
```

Define the panel item to accept user input for duration.

```
panel_items[DURATION_ITEM] =
  (Panel_item) panel_create_item( panel, PANEL_TEXT,
  PANEL_LABEL_STRING,      "duration (sec/10):",
  PANEL_VALUE_DISPLAY_LENGTH, 10,
  PANEL_VALUE_STORED_LENGTH,  10,
  PANEL_ITEM_X,            10,
  PANEL_ITEM_Y,            60,
  0  );
/* check for null pointer */
```

Define the button to actually ring the alarm. Notice the callback procedure, `ring_the_alarm`, is registered here.

```
rr_button_image = (Pixrect *) panel_button_image(panel,
                "Rock and Roll",
                0,
                0  ) ;

panel_create_item (panel, PANEL_BUTTON,
   PANEL_ITEM_X,              60,
   PANEL_ITEM_Y,              85,
   PANEL_NOTIFY_PROC,   ring_the_alarm,
   PANEL_LABEL_IMAGE,         rr_button_image,
   0  );
```

This is the end of the main loop: fit everything into the frame, and put it on the screen.

```
  window_fit(panel);
  window_fit(frame);
  window_main_loop(frame);
  exit(0);
}
```

This is the callback from the button. It stores values from the user (if any) into the alarm data structure, and rings the alarm.

Notice the `window_set` call. This is where the alarm is actually rung.

```
void
ring_the_alarm()
{
  int   duration_in_tenths;

  example_alarm.beep_num =
    (int) atoi(
    panel_get(panel_items[BEEP_ITEM], PANEL_VALUE));

  example_alarm.flash_num =
    (int) atoi(
    panel_get(panel_items[FLASHES_ITEM], PANEL_VALUE));

  duration_in_tenths =
    (int) atoi(
    panel_get(panel_items[DURATION_ITEM], PANEL_VALUE));

  example_alarm.beep_duration.tv_usec
    = (duration_in_tenths % 10) * 100000;

  example_alarm.beep_duration.tv_sec
    = (duration_in_tenths / 10);

 window_set(frame, WIN_ALARM, & example_alarm, 0);
}
```

Programmable Alarms with Help

Finally, consider the following code, which adds Spot Help to the programmable alarms example above.

We need to include `help.h`.

We also define a macro to use in entering `HELP_DATA`.

```
#include    <suntool/sunview.h>
#include    <suntool/panel.h>
#include    <suntool/help.h>
#define     BEEP_ITEM       0
#define     FLASHES_ITEM        1
#define     DURATION_ITEM       2
#define     NUMBER_OF_ITEMS     3

#define     P_ALARM_HELP(x)     HELP_DATA, "p_alarms:x"
void            ring_the_alarm();

Frame           frame;
Panel           panel;
Panel_item      panel_items[NUMBER_OF_ITEMS];
Win_alarm       example_alarm;
Pixrect *       rr_button_image;
```

This example uses a macro, `P_ALARM_HELP` to specify the `HELP_DATA` that Spot Help will use.

Note the `p_alarms` field in the macro; this directs Spot Help to look in a file named `p_alarms.info` in the help directory defined in the defaults database by Defaults Editor.

The other field in the macro, `x`, is a variable that is replaced with the key that Spot Help used to find the actual text in the file.

In the main loop we use the `P_ALARM_HELP` macro to indicate that we want to add help for the frame and the panel.

This directs Spot Help to the entries `:frame` and `:panel` in the `p_alarms.info` file.

```
int
main()
{
  frame = (Frame) window_create( NULL, FRAME,
    FRAME_LABEL,    "Programmable Alarms Example",
    P_ALARM_HELP(frame),
    0  );
  if (frame == (Frame) NULL ){
    fprintf(stderr, "SunView not available\n");
    exit(1);
  }

  panel =  (Panel) window_create( frame, PANEL,
    P_ALARM_HELP(panel),
    0  );
  /* check for null pointer */
```

Here we add help for each panel text item the user can enter, as well as the button.

The use of the macro is the same as in the previous figure.

```
  panel_items[BEEP_ITEM] = (Panel_item) panel_create_item(
    panel, PANEL_TEXT,
    PANEL_LABEL_STRING,     "beeps per alarm  :",
    PANEL_VALUE_DISPLAY_LENGTH, 10,
    PANEL_VALUE_STORED_LENGTH,  10,
    PANEL_ITEM_X,           10,
    PANEL_ITEM_Y,           10,
    P_ALARM_HELP(beeps),
    0  );
  /* check for null pointer */
  panel_items[FLASHES_ITEM] =
    (Panel_item) panel_create_item(
    panel, PANEL_TEXT,
    PANEL_LABEL_STRING,     "flashes per alarm:",
    PANEL_VALUE_DISPLAY_LENGTH, 10,
    PANEL_VALUE_STORED_LENGTH,  10,
    PANEL_ITEM_X,           10,
    PANEL_ITEM_Y,           35,
    P_ALARM_HELP(flashes),
    0  );
  /* check for null pointer */
```

More panel items

```
panel_items[DURATION_ITEM] =
   (Panel_item) panel_create_item(
   panel, PANEL_TEXT,
   PANEL_LABEL_STRING,      "duration (sec/10):",
   PANEL_VALUE_DISPLAY_LENGTH, 10,
   PANEL_VALUE_STORED_LENGTH,  10,
   PANEL_ITEM_X,            10,
   PANEL_ITEM_Y,            60,
   P_ALARM_HELP(duration),
   0  );
/* check for null pointer */

rr_button_image = (Pixrect *) panel_button_image(panel,
                  "Rock and Roll",
                  0,
                  0  ) ;

panel_create_item
   (panel, PANEL_BUTTON,
    PANEL_ITEM_X,               60,
    PANEL_ITEM_Y,               85,
    PANEL_NOTIFY_PROC,          ring_the_alarm,
    PANEL_LABEL_IMAGE,          rr_button_image,
    P_ALARM_HELP(rock_and_roll_button),
    0  );
```

Nothing needs to be added to the end of the main loop or to the callback.

```
  window_fit(panel);
  window_fit(frame);
  window_main_loop(frame);
  exit(0);
}
void
ring_the_alarm()
{
  int   duration_in_tenths;
  example_alarm.beep_num = (int) atoi(
    panel_get(panel_items[BEEP_ITEM], PANEL_VALUE));

  example_alarm.flash_num =  (int) atoi(
    panel_get(panel_items[FLASHES_ITEM], PANEL_VALUE));

  duration_in_tenths = (int) atoi(
    panel_get(panel_items[DURATION_ITEM], PANEL_VALUE));

  example_alarm.beep_duration.tv_usec
    = (duration_in_tenths % 10) * 100000;

  example_alarm.beep_duration.tv_sec
    = (duration_in_tenths / 10);

  window_set(frame, WIN_ALARM,  & example_alarm, 0);
}
```

Now, let's look at the file containing the Spot Help text.

the frame,

the panel,

```
:frame
Help for Programmable Alarms: frame

:panel
Help for Programmable Alarms panel

This program illustrates the use of programmable alarms,
new with SunOS 4.1.  They allow SunView programmers
to set the number of bells per alarm, the duration of the
bell, and whether each bell is audible, visible, or both.

This tool contains three text items to set the
parameters of the alarm, and a button that rings it.
```

the *beeps* panel text item,

```
:beeps
Help for Programmable Alarms: beeps

Enter an integer into this text item to control
how many times the alarm beeps.  If you don't
hear a beep, use the default editor to make
sure that the audible bell is enabled.
```

Next, we have the *flashes* panel text item,

```
:flashes
Help for Programmable Alarms: flashes

Enter an integer into this text item to control
how many times the alarm flashes.  If you don't
see a flash, use the default editor to make
sure that the visible bell is enabled.
```

the *duration* panel text item,

```
:duration
Help for Programmable Alarms: duration

This text item controls the duration of each
flash/beep of the alarm.  The units are tenths
of a second.  Enter an integer!  Non-zero!
```

the rock & roll button,

```
:rock_and_roll_button
Help for Programmable Alarms: rock_and_roll button

Hit this button to see and hear what your settings
do to the beeps, flashes, and duration.
```

and the end of the file.

```
:end_of_file
```

D.3. Colored Panel Items

SunView 1.80 offers the PANEL_ITEM_COLOR attribute to support colored panel items. Its use is simple:

```
PANEL_ITEM_COLOR, color,
```

The *color* should be given as an index into a colormap, such as is found in `sunwindow/cms_rainbow.h`.

Color Panel Example

In this example, a frame and panel are created with a variety of colored panel items.

Include the colormap header file.

The main loop.

`cms_rainbowsetup()` is a macro defined in `cms_rainbow.h`.

Set and name the colormap.

```
#include <stdio.h>
#include <suntool/sunview.h>
#include <suntool/panel.h>
#include <sunwindow/cms_rainbow.h>

Frame           frame;
Panel           panel;
Panel_item      orange_button, red_choice, indigo_toggle;
Panel_item      indigo_toggle, green_message;
Panel_item      green_message, blue_text, violet_slider;
Pixwin          *pw;
u_char          red[8], blue[8], green[8];
Pixrect *   button_image;

main()
{
  frame =  (Frame) window_create(NULL,FRAME, 0);
  if (frame == (Frame) NULL ) {
    fprintf(stderr, "SunView not available\n");
    exit(1);
  }
  panel =  (Panel) window_create(frame,PANEL, 0);
  /* check for null pointer */

  cms_rainbowsetup(red,green,blue);
  pw = (Pixwin *) window_get(panel,WIN_PIXWIN);

  pw_setcmsname(pw,"colorpanel");
  pw_putcolormap(pw,0,8,red,green,blue);
```

Now add an orange button panel item,

```
button_image = (Pixrect *) panel_button_image(
  panel,"Orange",0, 0);

orange_button = (Panel_item) panel_create_item(
  panel,       PANEL_BUTTON,
  PANEL_LABEL_IMAGE,  button_image,
  PANEL_ITEM_COLOR,  ORANGE,
  PANEL_ITEM_Y,    ATTR_ROW(3),
  PANEL_ITEM_X,    ATTR_COL(0),
  0);
/* check for null pointer */
```

a multiple choice in red,

```
red_choice = (Panel_item) panel_create_item(
  panel,       PANEL_CHOICE,
  PANEL_LABEL_STRING, "Red Choice",
  PANEL_CHOICE_STRINGS,   "one", "two", "three", 0,
  PANEL_ITEM_COLOR,  RED,
  PANEL_ITEM_Y,    ATTR_ROW(5),
  PANEL_ITEM_X,    ATTR_COL(0),
  0);
/* check for null pointer */
```

display a message in green,

```
green_message = (Panel_item) panel_create_item(
  panel,       PANEL_MESSAGE,
  PANEL_LABEL_STRING, "This is a Green message",
  PANEL_ITEM_COLOR,  GREEN,
  PANEL_ITEM_Y,    ATTR_ROW(7),
  PANEL_ITEM_X,    ATTR_COL(0),
  0);
/* check for null pointer */
```

some blue panel text,

```
blue_text = (Panel_item) panel_create_item(
  panel,       PANEL_TEXT,
  PANEL_LABEL_STRING, "Color: ",
  PANEL_VALUE,     "Blue",
  PANEL_ITEM_COLOR,  BLUE,
  PANEL_ITEM_Y,    ATTR_ROW(9),
  PANEL_ITEM_X,    ATTR_COL(0),
  0);
/* check for null pointer */
```

a toggle in indigo,

```
  indigo_toggle = (Panel_item) panel_create_item(
    panel,       PANEL_TOGGLE,
    PANEL_LABEL_STRING, "Indigo Toggle",
    PANEL_CHOICE_STRINGS, "one", "two", "three", 0,
    PANEL_ITEM_COLOR, INDIGO,
    PANEL_ITEM_Y,    ATTR_ROW(11),
    PANEL_ITEM_X,    ATTR_COL(0),
    0);
  /* check for null pointer */
```

and finally, a violet slider.

```
  violet_slider = (Panel_item) panel_create_item(
    panel, PANEL_SLIDER,
    PANEL_LABEL_STRING, "Violet Slider",
    PANEL_MIN_VALUE,    0,
    PANEL_MAX_VALUE,    10,
    PANEL_VALUE,    5,
    PANEL_ITEM_COLOR, VIOLET,
    PANEL_ITEM_Y,    ATTR_ROW(13),
    PANEL_ITEM_X,    ATTR_COL(0),
    0);
  /* check for null pointer */
```

The end of the program

```
window_main_loop(frame);
exit(0);
}
```

D.4. 24 Bit Color

The CG8 and CG9 frame buffers provide 24-bit *true color*, supported by the **Pixrect** and **SunView1** libraries. This section describes the **CG9** hardware and how it differs from previous Sun frame buffers. The subsequent sections explain how these differences are seen by an application programmer, and address compatibility issues with existing applications.

Additional Documentation

When reading this section, it may be useful to have read, or have available, the following manuals:

- *Pixrect Reference Manual*, for a detailed discussion of plane groups,
- *SunOS Command Reference Manual*, for `shelltool` and `cmdtool`,
- *SunView 1 Programmer's Guide*, for `ttysw`, `textsw`, and `panels`,
- *CG9 Release Notes*, for more specific information on the hardware.

Plane Groups

Like the **CG4**, the **CG8** and the **CG9** have three plane groups. There is a color plane group, which for the **CG8** and the **CG9** is 24-bits per pixel, and there is a monochrome overlay plane group with an associated overlay-enable plane group. The overlay is provided for fast monochrome performance of text windows.

The **CG8** and the **CG9** have an enhanced overlay/overlay-enable implementation compared to the **CG4**. A zero in the **CG4** overlay-enable causes the 8-bit plane group value for that pixel to be displayed rather than the overlay 1-bit value. The **CG8** and **CG9** requires both the overlay-enable **and** the overlay planes be zero to show the 24-bit color plane group value. The **CG8** and **CG9** thereby allow three overlay colors rather than the two available with the **CG4**. The two implementations are compared in the following table.

Table D-1 *Enable/Overlay Planes for CG4* and *CG8/CG9*

Overlay Plane	*Enable Plane*	*CG4*	*CG8/CG9*
0	0	8-bit color	24-bit color
0	1	color 0	color 1
1	0	8-bit color	color 2
1	1	color 1	color 3

Colormaps: Index Color vs. True Color Frame Buffers

Sun color frame buffers display at each pixel a 24-bit color value, defined by 8-bits (256 shades) of each of red, green, and blue. This yields 16.7 million different possible colors (2^{24}). However, previous frame buffers limit the number of different 24-bit colors that can be shown simultaneously.

The **CG4** column of Table A-1 refers to 8-bit color, color 0, and color 1. The 8-bit color value that is stored in the frame buffer's memory is actually an index into a color lookup table of 256 entries of 24-bit color values. For example, a pixel value of zero indicates to the frame buffer to display the 24-bit value contained at entry zero of the color lookup table. Additionally, the overlay has a two entry color lookup table associated with it.

The entries *color 0* and *color 1* in the table refer to 24-bit colors in the overlay color lookup table. Because different applications may desire a different set of colors selected from the 16.7 million different colors, methods for colormap changing, sharing, and swapping have been required. (See `pr_putcolormap` in the *Pixrect Reference Manual*, and `pw_setcmsname` and `pw_putcolormap` in the *SunView Programmer's Guide*.)

The **CG8** and **CG9** are *true color* framebuffers. Each pixel located in the **CG9** frame buffer's memory can hold an entire 24-bit color value. Therefore, indexing is not necessary and, although the **CG9** has a colormap, it serves a different purpose. The **CG9** colormap has 256 entries for each of red, green, and blue. These entries are changed only for gamma-correction of a color monitor.

Because `pr_putcolormap` and `pw_putcolormap` are frequently used in existing software, the semantics of these functions have been left intact and are ignored by the **CG9** with regard to the actual hardware color lookup tables.

However, recognizing that application programs might want to change the hardware color lookup tables, `pr_putlut` and `pr_getlut` commands have been created (*lut* is an abbreviation of *look-up table*). Likewise, the colormap commands have had a specific meaning to the overlay plane group and this meaning is unchanged, although the **CG9** has three colors rather than two in its overlay. `pr_putlut` and `pr_getlut` provide the new semantics in this case as well. The *CG9 Release Notes, Chapter 2* gives the differences among `pr_putcolormap`, `pr_getcolormap`, `pr_putlut`, and `pr_getlut`.

D.5. Double Buffering

Another hardware feature of the **CG9** is double buffering. (It is the double buffered version of the **CG8**.) Some **CG3** and all **CG5** frame buffers have two copies of the color frame buffer to allow double buffering. An application can write to one or both buffers while displaying the other, allowing for smooth animation because the viewer does not see the graphics creation. In the case of the **CG3** and **CG5**, both frame buffers are 8-bits deep and independently fit the same 8-bit scheme. The **CG9** accomplishes double buffering by splitting a 24-bit pixel into two 12-bit pixels. The application programmer reads and writes to each of the double buffers as if they were 24-bit, but the **CG9** hardware thresholds the color by storing only the high-order nibble of each of red, green, and blue.

While in double-buffer mode, an application may not read the same value back from the double buffer that was written to it.

`CANVAS_COLOR24` Attribute and Compatibility

Very little of the SunView API has changed; there is a new attribute:

```
CANVAS_COLOR24 , TRUE
```

This section explains the aspects of programming the **CG9** and **CG8** for the SunView application programmer.

A SunView application canvas defaults to monochrome unless a `pw_putcolormap` call is made to create an 8-bit canvas. This model is true for the **CG9** with slight variation. First, a new attribute has been added called `CANVAS_COLOR24`, which is set *TRUE* if the application wants to use *true color* 32-bit XBGR in the canvas. In this situation, all functions such as `pw_rop` and `pw_vector` work on 32-bit values in XBGR* format.

If the `CANVAS_COLOR24` attribute is not set and a `pw_putcolormap` call is not made, then the canvas defaults to the monochrome overlay.

In XGBR format, a 32-bit word is divided into four *channels* of 8 bits each. The X channel (the high-order 8 bits) is currently undefined and reserved for future enhancements. The next channel contains 8 bits for the blue color component. The other two channels hold corresponding information for the green and red components. The three components index the red, green, and blue portions of a look-up table, giving RGB components which combine to produce a particular hue and intensity that is seen on the screen.

8-Bit Color Mode

If a `pw_putcolormap` call is made, the canvas is placed in the 24-bit plane group but uses 8-bit indexed operations. In this situation, all functions such as `pw_rop` and `pw_vector` work on 8-bit indexed color values but display the appropriate 24-bit value. There are a few cautions associated with this mode of operation. One is that the actual depth of the canvas is 32-bits deep so operations to a memory pixrect have the same limitations as described in Table A-2 below. The standard *rop* operations between pixrects of different depths are allowed to some extent, as summarized in the table below.

Table D-2 *rop Operations (Limitations)*

Operation			Allowed?
0	→	***n***	yes
1	→	***n***	yes
n	→	***n***	yes
n	→	***1***	no
24	→	***32***	no
32	→	***24***	no

The value *n* can be 1, 8, or 32, but not 24 (bits). Note that 8-to-32 bit and 32-to-8 bit are not supported. To translate pixel colors between 8 and 32, use the formula shown below. This format uses the 8-bit pixel value (the variable `color8`) with the 8-bit colormap to generate a 24-bit color, which is saved in the integer variable `color24`. This `color24` variable has its true color stored in `XBGR` format. The value can then be saved as a 32-bit pixel in the pixrect's `PIXPG_24BIT_COLOR` plane group.

```
int color24;
unsigned char red[256],green[256],blue[256];
color24 = red[color8] +
    (green[color8] << 8) + (blue[color8] << 16);
```

Summary of 24 Bit Color Usage

The use of this attribute is summarized below.

Table D-3 *Color Attribute Usage Summary*

Effect	SunView Attributes
mono	window_create()
8-bit indexed emulation	window_create() pw_putcolormap *or* window_create(CANVAS_COLOR24 , TRUE) pw_putcolormap
24-bit	window_create(CANVAS_COLOR24 , TRUE)

Memory Pixrects

You can create 24-bit *memory* pixrects, which may be useful for synthesizing images that are later displayed.

It can be more efficient to use a 24-bit memory pixrect first to generate an image, and then to save that image as a 24-bit rasterfile. When `pr_load()` is called to load a 24-bit rasterfile, however, it automatically loads it as a 32-bit pixrect so that **Pixrect** operations run more efficiently. When `pr_dump()` is called, the converted pixrect is saved in a 32-bit rasterfile.

No double buffering in 8-bit indexed mode.

Another caution is that double buffering uses 24-bit to 12-bit thresholding, which tends to confuse the 8-bit indexed mode. Thus, double buffering is not supported in 8-bit indexed mode. Furthermore, because of the differences in hardware colormaps between 8-bit frame buffers and 24-bit frame buffers, colormap animation is also not supported.

Avoid duplicate colormap values in 8-bit indexed mode.

One final caution associated with 8-bit indexed mode is related to redundant colormap entries. If the application has multiple index entries with the same 24-bit color value, then some operations may fail because the wrong index might be used. This is easily overcome through minor changes to the colormap values.

When writing application programs, make sure that all entries in the colormap are unique. This action guarantees that reverse indexing from a true-colored pixel value back to the index value is correct. If several entries must share the same color, these entries can vary slightly on the lower bits, which typically does not result in any visual difference. For example, if four entries must have the same color of (255, 0, 155), do not initialize the colormap like this:

NOT THIS WAY

```
struct color {unsigned char r, g, b; } cmap[] = {
    { 255, 0, 155, }
    { 255, 0, 155, }
    { 255, 0, 155, }
    { 255, 0, 155,}};
```

Instead, initialize the colormap as follows:

THIS WAY

```
struct color {unsigned char r, g, b; } cmap[] = {
    { 255, 0, 155, }
    { 255, 1, 155, }
    { 255, 0, 156, }
    { 256, 0, 155,}};
```

Transparent Overlay

A new feature associated with the **CG9** is the ability to switch to the overlay plane or the overlay-enable plane from a color canvas, which allows quick rendering of text or graphics over the canvas without disrupting the underlying 24-bit image. This action is accomplished through a new **Pixwin**, call `pw_set_planes_directly`. This function takes three parameters:

1. The **Pixwin** pointer to the **Pixwin** of the canvas.
2. The plane group to which you wish to change.
3. The planemask associated with the new plane group.

Special caution should be taken to use `pw_lock` and `pw_unlock` around this code. Also, always restore the canvas to its original state before unlocking.

An example of the use of `pw_set_planes_directly` follows:

```
int plane_group_save, planes_save;

/* Pixrect *pw from canvas, be sure to call pw_lock */
/* save old state of canvas */

plane_group_save = pr_get_plane_group(pw->pw_pixrect);

(void) pr_getattributes(pw->pw_pixrect, & planes_save);

pw_set_planes_directly(pw,PIXPG_OVERLAY,1);

/* all pw functions now affect the overlay on the canvas */
/* restore old state of canvas before unlocking */

pw_set_planes_directly(pw, plane_group_save, planes_save);

/* unlock the pw region */
```

Note that the overlay-enable plane has a different definition than that for the **CG4**. The overlay colors in the overlay colormap shown in *CG9 Release Notes, Chapter 1* are set by SunView as follows:

Table D-4 *SunView Overlay Colors*

Overlay	*Overlay Enable*	*Color*
0	0	24-bit value
0	1	Window System Background Color
1	0	Window Foreground Color
1	1	Window System Foreground Color

The "1 0" color is set to the foreground color in order to have the cursor show up as the correct color over the canvas.

When using this feature be aware of the following:

- **Always** use `pw_lock(unlock)` when alternating between plane groups.
- **Always** return to the real canvas plane group before unlocking.
- SunView knows only about the real plane group of the canvas. Therefore, the overlay planes cannot be retained or redisplayed by SunView. Repairing damaged canvases is the responsibility of the application programmer.
- Since the cursor is the same color as the overlay foreground, it may disappear in regions filled with that color.
- Do not call `pr_putlut` from a **Pixwin** application. If you change the overlay colormap through a `pr_putlut` command in a **Pixwin** application, all overlay windows flash to the new color and the windowing system keeps restoring the real colors.

Cursor

Unlike the **CG4**, with the **CG8** and the **CG9**, the cursor is **always** in the overlay. Therefore, all cursor `rop` operations, such as exclusive OR's, are performed in the overlay plane and **never** in the 24-bit color plane.

Command Line Options

Unlike the **CG4**, neither the **CG8** nor the **CG9**, supports access to the monochrome overlay and the 24-bit color plane as two distinct desktops. Thus the following SunView command line options are disabled.

Figure D-1 *Disabled SunView Flags*

DISABLED - do not use with CG8 or CG9

```
%sunview -8bit_color_only
%sunview -overlay_only
%sunview -toggle_enable
```

Text subwindows in SunView tools such as `shelltool`, `cmdtool`, and `textedit` have command line arguments that allow you to specify a foreground and background color for a window. These command line options are as follows:

```
-Wf r g b -Wb r g b -Wg
```

(See `sunview`(1) in the *SunOS Command Reference Manual* for a definition of these options.) The **CG9** (unlike the **CG8**), supports all of these options, but performance declines when using `-Wg`, since every pixel of every character in the window requires 32-bit operation, instead of the 1-bit operation required if the window remained in the overlay.

D.6. Keyboard Support

A number of questions have arisen about the usage of the `.textswrc` file. These are addressed here.

There are 15 keys on the right hand side (the keypad) that can have functions assigned to them. Each key can be named:

```
KEY_RIGHT(n)      where 15 ≥ n ≥ 1
R(n)
Rn
```

Similarly, the top function keys each have three names:

```
KEY_TOP(n)            where 12 ≥ n ≥ 1
F(n)
Fn
```

The functions assigned to the keys are constructed from filters. When a function key is pressed with a text selection, the selected text is piped through the filter assigned to that key. The output is then piped back into the text at the carat. (If the selection was pending-delete, the original text is removed.)

There are a number of special filters, documented in `textedit_filters(1)`, that are provided especially for SunView users.

- `insert_brackets`,
- `remove_brackets`,
- `align_equals`,
- `shift_lines`.

Note, however, that any reasonable combination of shell commands can be used as a text subwindow filter.

A function is assigned to one of these keys by including in the `.textswrc` file a statement like:

```
/*
 * Note that:
 *  insert_brackets  /* */  does NOT work
 */
KEY_TOP(10) FILTER
insert_brackets "/* " " */"
```

This example shows how to include C language comment markers around a piece of text. You would enter this snippet into your `.textswrc` file, and save the

file. Then bring up a new `textedit` tool, since the changes only become effective when a tool is started. Then, select the text pending-delete, and press the key. The text will be replaced with a copy of itself surrounded with "/* " and "*/". Note the C-like syntax of comments in `.textswrc`.

The following example does the same thing, by entering the octal value for characters in the desired string.

```
KEY_TOP(10) FILTER
insert_brackets "\057*\040" "\040*\057"
```

You might want to add a filter to remove comments:

```
KEY_TOP(10) FILTER
remove_brackets "/* " " */"
```

Several filters shown below are handy for `troff` users.

For `troff` *italics*

Several `troff` command pairs, each on its own line

```
R(1)    FILTER
insert_brackets "\fI" "\fP"

KEY_TOP(12) FILTER
insert_brackets ".BS\n.LS\n" "\n.LE\n.BE"
```

The next group shows a variety of parentheses and quotes used:

parentheses,

quotes.

```
/*
 * Note: insert_brackets "("  ")" also works, and
 *   insert_brackets ( )     also works
 */
KEY_RIGHT(4)    FILTER
insert_brackets \( \)
/*
 * Note: insert_brackets "\""  "\"" does NOT work
 */
KEY_RIGHT(5)    FILTER
insert_brackets \" \"

/*
 * Note: insert_brackets "\'\'" "\'\'"  does NOT work
 */

KEY_RIGHT(9)    FILTER
insert_brackets  \'\' \'\'
```

The final example uses the `ls` command to obtain the listing of the current directory and pipe it into the text subwindow after a little formatting.

```
KEY_TOP(10) FILTER
ls | insert_brackets "*List**\n" "*End**" | shift_lines 4
```

This can be done with `awk` or `sed` scripts to great advantage.

```
KEY_TOP(10) FILTER
awk -f ~me/mydirectory/myscript
```

Limits to Assigning Keys

There are a number of restrictions on the use of function keys in SunView. The following keys are not assignable at all:

- [F1] is CAPS_LOCK
- [L1] is Stop, and also used with in the abort sqeuence.

Another group of keys cannot be mapped via `.textswrc` unless you set the `/Input/Arrow_Keys` default to "No"

- Keys [R8], [R10], [R12] and [R14] are the arrow keys;
- [R7] is `Home`, which moves to the beginning of the editing buffer,
- [R13] is `End`, which moves to the end of the editing buffer, and
- [R11] is the function `GO_LINE_FORWARD`, move to the start of next line.

The keys on the left keypad are not mapped directly. If the user sets the `/Input/Lefthanded` default to "Yes", the SunView functions move from the left keypad to the right one, and selected keys assigned to the right keypad appear on the left one.

D.7. Programming Hints

This section offers tips and techniques that are either not previously mentioned in the documentation or that concern how to handle bugs or known problems.

Memory Leaks From Button Images

A SunView program may fail to reclaim memory after destroying an object. This loss of useable memory ("memory leakage") is cumulative and eventually causes the system to crash. To avoid these problems, there are precautions to observe.

A button image is a separate object, and therefore is not destroyed with the button, (and may be reused, for example, with another button). Thus, the memory allocated by `panel_button_image()` for the panel button image is not freed when the panel button is destroyed.

To avoid this leak, create the button image explicitly, so that it has a handle by which it can be destroyed. The examples in *Color Panel Example* earlier in this Appendix show how this is done.

Counting File Descriptors

SunView makes heavy use of file descriptors, one result of which is that it is often useful to know how many file descriptors are open. This can be accomplished with the `fstat(2)` system call. The method is to loop over each possible `fd`, explicitly checking its status with `fstat(2)`.

The question of the upper limit of the loop can be answered either by choosing a suitable number such as 256, or more dynamically by using the `getdtablesize(2)` system call to determine the limit.

An example appears in the *Kernel Interface* chapter of the *System Services Overview*.

File Descriptor Leakage

`window_return` does not destroy the windows in addition to exiting from `window_loop`.

Some programmers may not realize that `window_return` exits from `window_loop`, but *does not destroy any windows*. As a result the file descriptors associated with the windows remain in use and unavailable for other windows. To reclaim those file descriptors, be sure to call `window_destroy`. Failure to use `window_destroy`, will cause error messages such as:

```
pr_open: open failed for /dev/fb
no more windows available
WIN ioctl number 1c: Too many open files
window: Window creation failed to get new fd
/dev/win49 would not open (be created)(errno = 24)
no more windows available
WIN ioctl number 1c: Too many open files
window: Window creation failed to get new fd
Segmentation fault (core dumped)
```

Null Pointers

Problems can arise when a SunView function call to create an object (frame, panel, or panel_item for example) returns `NULL`. You cannot blindly use such a pointer without first checking whether it is `NULL`. Although it is common practice not to check pointers, and usually does not create problems, it is careless programming and can lead to trouble.

The examples in this Update Appendix check for `NULL` pointers when creating base frames, and indicate by comments when to do so after the creation of other objects.

You are advised to adopt this practice in your own SunView code (and elsewhere too).

pixwin and pixrect

Pixwin calls (`pw_*`) offer a higher level of functionality than pixrect (`pr_*`) calls, and thus should be used whenever possible. Sometimes it happens that pixwin does not offer necessary functionality. In such cases, the pixrect interface is available, as defined in the documentation. However, the pixrect interface is more likely to change in the future than is pixwin. Moreover, undocumented calls are not supported, and should not be used.

Limitations of `icon_load_mpr()`

The present default settings for the file format parameters of icons are:
Format_version = 1

```
Width = 64,
Height = 64,
Depth = 1,
Valid_bits_per_item = 16.
```

These values are currently subject to the following limitations:

`Format_version` must always be 1.

`Width` must be a multiple of 16.

`Depth` must always be 1.

`Valid_bits_per_item` can only be 16 or 32.

Hardware for Multiple Desktops

To run multiple desktops on a single screen, the user needs a CG4 framebuffer (10 bit planes).

`ws_set_favor` Default Value Changed To 0.

The `ws_set_favor` flag controls whether or not the window driver will try to boost the priority of the window process (and its children) that has the current event lock. The default is 0. In very tight memory situations, setting this to 1 will improve interactive performance.

`TEXTSW_WRAPAROUND_SIZE` attribute

The attribute `TEXTSW_WRAPAROUND_SIZE` in the text subwindow package is not documented. It is of type `int` and specifies the maximum allowed size (in bytes) of the edit log file (in `/tmp`) associated with a text subwindow. The lower bound of this attribute is 8096, which is silently enforced. The default value is `TEXTSW_INFINITY` (allow the edit log file to grow as much as needed).

`notify_flush_pending`

`notify_flush_pending` removes (flushes) all pending events for a client. If you call it after doing a `window_destroy`, the destroy event is removed and the window will not be removed at all.

Interposing Scroll Handlers

When trying to interpose your own scroll handler, do not use `scrollbar_scroll_to` in the interposed routine, as it causes an infinite loop by generating another scroll event. The proper approach is to have the interposed routine either set a flag, set a timer, or generate a secondary non-scrolling event to be processed outside of the event handling pipeline.

Additional `auto_sigbits`

The following bits in `sigbites_ptr` should be noted by those writing their own prioritizers:

- `SIGTSTP` means `notify_destroy` should be called with status of `DESTROY_CHECKING`.
- `SIGTERM` means be`notify_destroy`should of `DESTROY_CLEANUP`.

- SIGKILL means notify_destroy should be called with status of DESTROY_PROCESS_DEATH.

FBIONREAD

There is no FBIONREAD SunView function; references to it result from a typographical error in the name of the correct call, FIONREAD.

FRAME_SHADOW and FRAME_SHOW_SHADOW incorrectly documented

Programs using FRAME_SHADOW instead of FRAME_SHOW_SHADOW to set or inhibit a shadow, will create an error message but continue execution. All subwindows will have shadows.

not all pixwin functions are documented

Not all the pixwin functions in /usr/include/sunwindow/pixwin.h are documented in either the *SunView Programmer's Guide* or the *SunView System Programmer's Guide*. Undocumented calls are not supported and are subject to change without notice.

SCROLL_NORMALIZE attribute

The default for SCROLL_NORMALIZE is TRUE. When scrollbars are used within panels, the default behavior is to scroll to the first line in view for a panel item. This can sometimes cause problems when trying to view a panel item, such as a choice item layed out vertically, since all PANEL_CHOICE_STRINGS may not be visible within the scroll region and cannot be scrolled into view because SCROLL_NORMALIZE is TRUE. In those instances set SCROLL_NORMALIZE to FALSE, in addition to setting the scrollbar's SCROLL_LINE_HEIGHT. For example:

```
#include <suntool/sunview.h>
#include <suntool/panel.h>
#include <suntool/scrollbar.h>
Frame       frame;
Panel       panel;
Panel_item  choice_item;
static int  choice_count = 0;
Scrollbar   sb;
main(argc,argv)
int     argc;
char    **argv;
{
frame = window_create(NULL,FRAME,
0);
sb = scrollbar_create(SCROLL_NORMALIZE, FALSE,
SCROLL_LINE_HEIGHT, 5,
0),
panel = window_create(frame,PANEL,
WIN_ROWS, 5,
WIN_VERTICAL_SCROLLBAR, sb,
0);
choice_item = panel_create_item(panel,PANEL_CHOICE,
PANEL_LABEL_STRING,  "Choices:",
PANEL_LAYOUT, PANEL_VERTICAL,
PANEL_CHOICE_STRINGS,
"01", "02", "03",
"04", "05", "06",
"07", "08", "09",
"10", "11", "12",
"13", "14", "15",
0,
0);
window_fit_height(frame);
window_main_loop(frame);
}
```

Subframes Cannot Be Iconified

Subframes (not subwindows, but frames created by calling `window_create( baseframe, FRAME)`) cannot be iconified. All subframes are intended to be transient and are not allowed to close to an icon.

FRAME_INHERIT_COLOR behavior

If you set the `FRAME_INHERIT_COLORS` attribute to `TRUE` for subwindows before setting up a colormap for the frame, the subwindows will not inherit the colors of the frame. The correct procedure is to set the attribute *after* the colormap.

THIS WAY

```
my_set_colormap_function();
window_set(frame, FRAME_INHERIT_COLORS, TRUE, 0);
```

NOT THIS WAY

```
window_set(frame, FRAME_INHERIT_COLORS, TRUE, 0);
my_set_colormap_function();
```

Destroying A Window Without Returning To The Notifier

The following code shows how to destroy a window without returning to the notifier. `window_destroy` posts to the window a destroy event which won't be processed until the notifier resumes.

Note that the call to `notify_flush_pending` may be necessary to remove pending events for the window.

```
destroy_subframe(sub_frame)
/* window_destroy(sub_frame) immediately */
Frame sub_frame;
{
    if (notify_post_destroy(sub_frame,
    DESTROY_CHECKING,
    NOTIFY_IMMEDIATE)
    != NOTIFY_DESTROY_VETOED) {
    (void)notify_flush_pending(sub_frame);
    (void)notify_post_destroy(sub_frame,
    DESTROY_CLEANUP,
    NOTIFY_IMMEDIATE);
    }
}
```

Using `window_create` without Error Message

Occasionally an application will want to call `window_create` repeatedly and yet not have it be apparent to the user when the system runs out of `/dev/win` devices. Either they wish to report the error themselves, or they just want to create as many windows as they can without making the user see an error message when the limit is reached.

Currently `window_create` outputs an error message to standard error when there are no more windows and returns `NULL`. So when the program finds out there is an error, the user has already seen the error message.

Currently, the only way around this is to redirect standard error away from the console.

.sunview and Environment Variable Expansion

There are limits on how shell environment variables in the `.sunview` file are expanded. For example, using paths like `$PROGDIR/file_to_run` or `~user/bin/file_to_run` does not work. For efficiency, SunView uses a simple `exec()` on each line in the file, so expansion does not occur. This is not likely to change in the future.

However, you can use variables like this if the command line is a call to `csh`:

```
csh -c $PROGDIR/file_to_run
csh -c ~flake/bin/file_to_run
```

Tools Off Sreen

Sometimes, when tools come up, the bottom edge of the tool will be partially off screen. This causes scrolling problems and/or funny characters. The effect of this problem can be alleviated by moving the window up so that it does not extend over the bottom of the screen.

`pw_putattributes`

`pw_putattributes` does not set the attributes of the retained memory pixrect. Also, bitplane masks do not work on a memory pixrect. These capabilities are necessary to do plane manipulation, for example, double buffering.

The following code can be used to create this type of memory pixrect:

```
#include <sys/types.h>
#include <pixrect/pixrect.h>
#include <pixrect/pr_util.h>
#include <pixrect/memvar.h>
Pixrect *
mem_create_with_planemask(w, h, depth)
int w, h depth;
{
        Pixrect *pr;
    struct mprp_data *mprd;
    if (pr = mem_create(w, h, depth))
        if (mprd = alloctype(struct mprp_data)) {
            mprd->mpr = *mpr_d(pr);
            free(mpr_d(pr));
            pr->pr_data = (caddr_t) mprd;
            mprd->mpr.md_flags |= MP_PLANEMASK;
            mprd->planes = ~0;
            }
    else {
        pr_destroy(pr);
        pr = 0;
        }
    return pr;
}
```

A sample call to this function might look something like this:

```
/*
 *      Fixes a retained memory pixrect in a canvas
 *      pixwin by replacing it with a memory pixrect
 *      which supports attributes.
 */
static void
fix_retained_pixrect(win,pw)
Window  win;
Pixwin  *pw;
{
  Pixrect *pr;
  int     w,h,d;
  w = (int)window_get(win,WIN_WIDTH);
  h = (int)window_get(win,WIN_HEIGHT);
  d = 8;
  if ((pr = mem_create_with_planemask(w,h,d)) == NULL) {
        fprintf(stderr,"Could not create memory pixrect0);
    exit(1);
  }
  pr_destroy(pw->pw_prretained);
  pw->pw_prretained = pr;
}
```

Filename Completion

Using the Esc key to obtain filename completion does not work in SunOS release 4.1.

Sticky Secondary Selections

Sunview sometimes gets stuck in secondary selection mode (when things are underlined). To correct this, go to a command window and use the `clear_functions` command. This should cause the selection service clients to give up their selections, thereby clearing the selection service.

Summary of SunView 1.80 Bug Fixes

This section consists of a table describing each bug or RFE (Request for Enhancement) that has been fixed in SunView 1.80, collated in ascending order by the Bug ID number (*Bugid*). The Bugid is the "reference number" that is assigned to each bug when it is reported, and is used subsequently to refer to it.

Table D-5 *Sunview 1.80 Fixed Bugs*

bugid	Summary Description
1002377	window display lock broken when window exposed
1002411	can start suntools when not at display
1002523	textedit incorrectly sizes windows with the `-Ww` flag
1002759	a frame created with `WIN_SHOW TRUE` gives `WIN ioctl` error
1003340	unnecessary repaint on upper split view
1003354	a blank `FILTER` entry in `.textswrc` hangs when key pressed
1003383	Prevent multiple inclusion of system include files
1003571	icon's default font not set until display time
1003581	`window_set FRAME_CLOSED` to `TRUE` gives "win ioctl" message

Table D-5 *Sunview 1.80 Fixed Bugs— Continued*

bugid	Summary Description
1003648	`lockscreen` does not accept passwords with control characters.
1003788	`image_browser_2` takes up (infinite?) CPU time on "Browse"
1003815	panel starts interval timer without keyboard focus
1003850	`Too large a window number (128)!` message with two framebuffers
1003877	signal TTIN while reading from standard input in dbx/scripts
1004221	Inverted suntools do not work on Prism's b/w framebuffer.
1004442	shelltool (and other tools) dump core when `$WINDOW_PARENT` not set
1004580	`textsw_file_lines_visible()` returns incorrect values
1004586	Multiple colors for panel items
1004838	ttysw uses `fork()` rather than `vfork()`
1005102	memory leak in `panel_destroy_item()`
1005499	tty subwindow goes into infinite loop if `open /dev/ttyp?` fails
1005729	`WIN_FONT` text subwindow attribute ignored
1006084	ttysw children do not exit
1006159	`cmdtool` does not allow `Reset` immediately after `Store`
1006173	damaging `suntool`'s colormaps can cause "panic: bus error"
1006217	`suntools` does not warn you it's busy when starting up
1006222	`defaultsedit` does not allow setting `nosunview`
1006341	`pw_line` draws different than `pw_vector` given same coordinates
1006426	`window_create` failed to return `NULL` on failure for panel
1006560	Can not use ~ to specify icon in SunView `.rootmenu` file
1006591	exit of suntools does not zero out `/etc/utmp` entries
1006652	`screenblank` dumps core if argument is missing
1007187	`shelltool` dumps core if given -Wt with no arguments
1007442	`rectlist.h` needs guard
1007443	`PW_DBL_WRITE` sets to `PW_DBL_FORE` when creating menu
1007620	`screenblank` persists despite keyboard activity
1007696	`suntool` should return windowfd when using lint in llib-l
1008155	Find and Replace loops replacing single occurrence of text
1008199	toggling `TTY_ARGV` fails
1008457	panels do not go to back on Shift-L5
1008564	Cutting selected text in subwindow leaves selection on-screen
1008579	`textedit` scrollbar bubble misplaced for very large files
1008849	faulty SunView program gives "panic: bus error"
1008949	`cmdtool` dumps core after bad option on command line.
1009260	tty windows fail to deallocate old frame icon pixrects
1009284	`notify.h` uses `fd_set` but does not include its definition
1009354	resizing window gives bad message
1009357	fullscreen resize option available when already fullscreen
1009507	`LOC_RGNEXIT` missing when scrollbars are south and east
1009822	second ghost caret from `mailtool` Compose=>Include
1010462	text subwindow still uses old menu_prompt instead of alerts
1010463	CTRL-tab in a text subwindow does not work after a tab is pressed
1010485	Find=>Selection Forward does not work in `mailtool`

Table D-5 *Sunview 1.80 Fixed Bugs— Continued*

bugid	Summary Description
1010522	`shift_lines` will not shift left
1010557	`Save layout` uses current directory, not home directory
1010746	File=>Load File accelerator works too often
1010847	`shift_lines -t -1` broken, also Text=>Extras and `.textswrc`
1010972	Pull right menu appears,then vanishes mysteriously
1011026	double-buffering and `pw_text` interact badly
1011042	`MENU_GEN_PROC` creates bogus `MENU_STRING_ITEM`, value pair
1011090	must have a `.defaults` file
1011133	`shift_lines` requires `.indent.pro` file
1011234	illegal memory free in suntools
1011384	file truncated in `textedit` upon save with full filesystem
1011412	alert attribute `ALERT_BUTTON_YES` disables right mouse menu
1011519	suntools hangs with CG3 and GP1 or GP+
1011853	Exiting suntools without quitting chesstool causes machine to panic
1011938	`PW_DBL_EXISTS` not defined in `<sunwindow/pw_dblbuf.h>`
1011973	Sun-2 and Sun-3 shared library version numbers are different
1012020	textsw filters drop data
1012023	lockscreen security hole
1012414	-Wf flag for tools broken
1012415	cursor not in foreground color
1012448	`cat` a binary in `cmdtool` results in a core dump.
1012506	`PANEL_ITEM_BOXED` in panel.h but not implemented
1012577	`textsw_insert` gets progressively slower in 3.x
1012580	scroll buttons do not work in canvas_demo
1012587	creating a tty after one has been destroyed hangs program
1012695	tools started from SunView sometimes do not appear
1012757	tty subwindow calls `signal(3)` which is illegal in SunView
1012758	ESC[0;7m does not invert in shelltool on 3.x
1012774	menus cause intermittant colormap flashing in 4.0
1013745	in `textedit`, a Ctrl-Delete enters undeletable character
1013746	Get/Put from cmdtool to `textedit` window can kill cmdtool.
1013767	variable collision causes no cursor in ttysw when using FORTRAN77
1013901	`sunview -i` and bad root menu file results in all black screen
1014035	8-bit characters not displayed correctly when batching on
1014075	monochrome pixwin inheriting colormap of overlying cursor
1014145	`window_set()` should return non-zero value for success.
1014179	`textedit` dies with SEGV on blink owner
1014194	memory leak everytime a window is destroyed in notifier
1014751	arguments to `gfxsw_select()` defined incorrectly in `gfxsw.h`
1014820	Incorrect use of `select` system call in `win_bell.c`
1014884	changing the monochrome colormap in 4.0 does not work correctly
1014935	Debugging messages in sunwindowdev waste bytes
1015097	`attr.h` does not conform to ANSI C standard
1015167	52 menu items gives "menu_show: Menu too large for screen"

Table D-5 *Sunview 1.80 Fixed Bugs— Continued*

bugid	Summary Description
1015181	`icon_load_mpr` returns incorrect data.
1015394	`MENU_REPLACE_ITEM` causes memory leak,`menu_destroy` no effect
1016307	`attr_rc_units_to_pixels` does not traverse embedded av-lists
1016479	`FRAME_ICON` does not reclaim memory
1016585	`cframedemo` dumps core under 4.0
1016660	attribute header file `attr.h` uses but does not define `caddr_t`
1016718	RFE for editable panel text items
1016820	`ntfy_errno_abort_init` never gets set
1016876	The -R usage for SunView libraries is incorrect for 4.0
1017411	8-bit fonts are "eaten" by mouse if written on top of cursor
1017503	obsolete directory `/usr/lib/keymaps` in the distribution tape
1017814	panel will not be displayed correct on machine with GP2
1017887	`MENU_GEN_PROC` causes bus error
1018216	`win_enumall()` procedure causes errors when 128 windows exist
1018599	in FrameMaker, F6 and F7 operate incorrectly
1018720	`pw_line` does not rasterize patterned lines consistently
1018784	cursor color incorrect
1018785	dragging a window off the screen panics the kernel.
1018963	problem with `ptys` being left in bad state
1018992	selection destroys window and leaves `tty` device in bad state
1019086	`pctool: seln_Create failed` message on 4/330
1019256	macros not defined in `values.h`
1019290	`set_cursor` caused a canvas repaint
1019359	cannot get `PANEL_VALUE_FONT` of a panel text item
1019398	`textedit` allows user to create file with whitespace in name
1019404	request for more than 128 windows
1019664	`SunView: No such file or directory` message; missing `/dev/*`
1019891	SPARC cursor performance problem
1019897	bad cursor image size test in `winio_getusercursor()`
1019907	premature crosshair cursor pixrect image data allocation
1020222	`textsw_reset()` does not free memory
1020319	`LOC_WINEXIT` events are being delivered erroneously
1020397	`Too large a window number (128)!` message built into kernal
1020454	`cmdtool` can exit leaving pty in unusable state
1020719	`window_create` of subframe can crash instead of return `NULL`
1020730	process's `nice` value gets zeroed
1021270	escape sequences hang a `cmdtool` that is remote logged in.
1021476	`makefile` does not make `shelltool` or `cmdtool` target
1021477	icon code incompatible with pixrect library
1021544	pw_line does not work with retained, color canvases
1021664	`win_getnewwindow()` does not always return -1 failure
1022020	shift-L5 does not put windows to the back, and moves others to front
1022552	`suntools -s` with nonexistent filename will hang the system.
1022841	the edit log wraparound size has no effect on `cmdtool`

Table D-5 *Sunview 1.80 Fixed Bugs— Continued*

bugid	Summary Description
1023098	subframe destruction results in 4K memory leak
1025077	`cron` cannot execute the command `lockscreen`
1025689	Changing fonts in a tty will crash a sunview application
1025804	canvas damage is incorrectly cleared to red
1025886	lockscreen -e should log the user out
1026368	error messages from `ttysw_fork_it()` and `ttysw_tty_restore()` confused
1026613	`Resize->Fullscreen` dumps core
1026708	`textsw` edit back char botch
1026733	pressing pop-up menu button causes segmentation fault in sundiag
1026817	diamond (Meta) key on type-4 keyboard does not work under SunView
1026818	SunView colormap segmentation broken
1026820	cannot place caret after last character in full panel text item
1026936	`pixwins` does not alow non-power of 2 colormaps
1027435	missing file `sunwindow/cms_colorcube.h`
1027565	suntools.c has very poor security
1027642	No makefile in /usr/demo/SUNVIEW/SRCS/DATA.
1027650	first tool in SunView does not accept color
1027956	`strdup` redefined in `<subwindow/sun.h>` producing syntax error
1028029	Escape sequences botched by `cmdtool`
1028055	typed text in colored panel text items is wrong
1028073	`libsuntool make install_h` misses header files
1028230	linking dynamically with `libsunwindow` causes SunWrite core dump
1028260	8 bit emulation not working in SunView
1028299	CDROM and FACES makefiles missing dependencies
1028366	Cursor disappears in background during fullscreen mode
1028588	creating popup window crashes SunView tool
1028682	double click does not work properly in editable panel text items
1028685	caret not placed properly in editable panel text items
1028688	The default for pending delete is TRUE in editable panel text items
1028689	CTRL key toggle of `Adjust_is_pending_delete` fails in panel text item
1028772	`CANVAS_COLOR24` attribute generates spurious error message on CG4
1029033	Setting `WIN_SHOW` to `TRUE` causes crash
1029088	sundiag pop-up window does not accept any input
1029598	`swin (-g)` gives `win_get_focus_event: Error 0`
1029616	Internal API change can cause 4.*x* compatibility problems

Keyword Summary of Fixed Bugs

This section consists of an alphabetical list of keywords with the corresponding bug(s) whose fix(es) concern that keyword.

Table D-6 *Keyword Index to Fixed Bugs*

keyword	Relevant Bugs
1027565	
.defaults	1011090
.indent.pro	1011133
.rootmenu	1006560
.textswrc	1003354, 1010847
/dev/*	1019664
/dev/ttyp	1005499
/etc/utmp	1006591
/usr/lib/keymaps	1017503
8-bit characters	1014035
8-bit emulation	1028260
8-bit fonts	1017411
ALERT_BUTTON_YES	1011412
CANVAS_COLOR24	1028772
CDROM	1028299
CG*	1011519, 1028366, 1028772, 1025804
CTRL characters	1003648, 1010463, 1013745, 1028689
Escape sequences	1012758, 1021270, 1028029
FACES	1028299
FORTRAN	1013767
FRAME_ICON	1016479
Framemaker	1018599
GP	1011519, 1017814
LLOC_RGNEXIT	1009507
LOC_WINEXIT	1020319
MENU_GEN_PROC	1017887
MENU_REPLACE_ITEM	1015394
Meta key	1026817
PANEL_ITEM_BOXED	1012506
PANEL_VALUE_FONT	1019359
PHIGS	1019290
PW_DBL_EXISTS	1011938
SEGV	1014179, 1026733
SPARC	1019891
Sun-2	1011973
Sun-3	1011973, 1017411
Sun-386i	1017411
Sun-4	1017411, 1019086
SunWrite	1028230
TTIN	1003877
WIN_FONT attribute	1005729
WIN_SHOW	1002759, 1029033
`MENU_GEN_PROC`	1011042
`icon_load_mpr`	1015181

Table D-6 *Keyword Index to Fixed Bugs— Continued*

keyword	Relevant Bugs
`lockscreen`	1012023, 1025886
`pw_line`	1021544
`sundiag`	1026733, 1029088
`suntools`	1002411, 1027565
accelerator	1010746
adjust_is_pending_delete	1028689
alert	1010462, 1011412, 1028366
attr.h	1015097, 1016660
attr_rc_units_to_pixels	1016307
av-lists	1016307
background	1028366
browse	1003788
bus error	1006173, 1008849, 1017887
caddr_t	1016660
canvas	1006173, 1009507, 1012580, 1017411, 1019290, 1025804, 1021544
caret	1009822, 1013767, 1026820, 1028685
cmdtool	1006159, 1008949, 1012448, 1012695, 1013746, 1020454, 1021270, 1021476, 1022841, 1028029
cms_colorcube.h	1027435
color	1004586, 1012414, 1012415, 1018784, 1027650, 1028055, 1021544
colorcube	1026936
colormap	1006173, 1014075, 1014884, 1026818, 1026936, 1012774
compatibility	1029616
confirmer	1011412, 1028366
console	1002411
core dump	1004442, 1006652, 1007187, 1008949, 1012448, 1016585, 1026613, 1028230
cpu time	1003788
crash	1020719, 1028588, 1029033, 1025689
cron	1025077
cursor	1012415, 1013767, 1014075, 1017411, 1018784, 1019891, 1019897, 1019907, 1028366
defaultsedit	1006222, 1022841
double buffer	1007443, 1011026
drawing	1006341
edit log	1022841
environment variables	1004442
events	1020319
fd_set	1009284
flags	1002523, 1012414
fonts	1003571, 1017411, 1025689
foreground	1012414
fork	1004838, 1008199
frame	1002759, 1009260

Table D-6 *Keyword Index to Fixed Bugs— Continued*

keyword	Relevant Bugs
frame closed	1003581
framebuffer	1020319, 1004221
framebuffers	1003850
fullscreen	1009357, 1026613, 1028366
gfxsw.h	1014751
gfxsw_select	1014751
hang	1003354, 1021270, 1022552
icon	1003571, 1009260, 1021477, 1015181
include files	1003383, 1009284, 1027956
infinite loops	1005499
install_h	1028073
itimer	1003815
keyboard	1007620, 1026817
keyboard focus	1003815
library	1007696, 1011973, 1016876, 1021477, 1028073, 1028230
lint	1007696
lockscreen	1003648, 1025077
mailtool	1009822, 1010485
makefile	1021476, 1028073, 1028299, 1027642
memory	1011234
memory leak	1005102, 1014194, 1015394, 1016479, 1020222, 1023098
menu	1007443, 1010972, 1011412, 1015167, 1026733, 1028366, 1012774, 1011042
menu_destroy	1015394
menu_prompt	1010462
message	1003850
messages	1002759, 1003581, 1009354, 1014935, 1019086, 1019664, 1020397, 1029598
mouse	1011412, 1017411, 1028366
nice	1020730
notifier	1014194, 1029088
notify.h	1009284
ntfy_ermo_abort_init	1016820
panel	1003815, 1004586, 1006426, 1008457, 1017814
panel text	1016718, 1019359, 1026820, 1028055, 1028682, 1028685, 1028688, 1028689
panel.h	1012506
panel_destroy_item()	1005102
panel_item	1012506
panic	1006173, 1011853, 1018785
passwords	1003648
pctool	1019086
performanc	1019891
pixfont	1025689

Table D-6 *Keyword Index to Fixed Bugs— Continued*

keyword	Relevant Bugs
pixrect	1009260, 1019907, 1021477
pixwin	1006173, 1014075, 1026936
pty	1018963, 1020454
pw_batch_*	1014035
pw_dbl_for	1007443
pw_dbl_write	1007443
pw_line	1006341, 1018720
pw_putcolo	1006173
pw_text	1011026, 1014035
pw_vector	1006341
reboot	1011519
rectlist.h	1007442
remote login	1021270
repaint	1003340, 1019290
resize	1009357
root menu	1013901
save layout	1010557
screenblank	1006652, 1007620
scroll buttons	1012580
scrollbar	1008579, 1009507
security	1002411, 1012023, 1027565
select system call	1014820
selection	1008564, 1010485, 1010972, 1018992
seln_create	1019086
set_cursor	1019290
shelltool	1004442, 1007187, 1009260, 1012695, 1012758, 1021476
shift_lines	1010522, 1010847, 1011133
signal(3)	1012757
split view	1003340
standard input	1003877
strdup	1027956
subframe	1020719, 1023098
subwindow	1012757
sun-3	1025689
sun-4	1025689
suntools	1006173, 1006217, 1006560, 1006591, 1007696, 1010557, 1011234, 1011519, 1011853, 1012695, 1022552, 1004221
sunwindowdev	1014935
swin	1029598
textedit	1002523, 1008579, 1011384, 1013745, 1013746, 1014179, 1019398
textsw	1003354, 1005729, 1008155, 1008564, 1010462, 1010463, 1010485, 1010847, 1020222, 1022841, 1026708, 1012020
textsw_file_lines_visible()	1004580
textsw_insert	1012577

Table D-6 *Keyword Index to Fixed Bugs— Continued*

keyword	Relevant Bugs
textsw_reset	1020222
toolplaces	1010557
tty	1003877, 1012587, 1012757, 1018992, 1025689
tty_argv	1008199
ttysw	1004838, 1005499, 1006084, 1009260, 1013767, 1026368
values.h	1019256
vfork	1004838
win_bell.c	1014820
win_enumall	1018216
win_getnewwindow()	1021664
window	1013746, 1014194, 1014935, 1018216, 1018785, 1018992, 1019404, 1022020, 1028588
window display lock	1002377
window_create	1006426, 1020719
window_set	1003581, 1014145
windowfd	1007696
windows	1009260, 1009354, 1011519, 1029088
winio_getusercursor()	1019897
wraparound	1022841

Index

S

T

W