D
Version 3.2 and the File Chooser

The XView *File Chooser* is an implementation of the OPEN LOOK Application File Choosing Specification. The Application File Choosing Specification supercedes the traditional OPEN LOOK Style Guide on the subject of opening and saving files from an application, without the use of the File Manager. The file chooser also lets the user easily navigate through the file system. Note that the file chooser is not intended to replace the File Manager; any application supporting the file chooser is also required by the File Choosing Specification to support drag and drop with the File Manager, in accordance with the OPEN LOOK Drag and Drop Specification.

File choosers provide a simple and consistent interface for opening and saving files to the UNIX File System. File choosers also provide a Go To Menu for going to different directories which allows the user to easily select a directory from a predefined list, an application defined list or a dynamicly created history list. Normally a file chooser will be presented from an application's File Menu. The user will then have selections for opening a document, saving a document, or saving a document to a new name. Once one of these selections is made, a file chooser object is presented.

D.1 Creating File Choosers

To use the `FILE_CHOOSER` package, include the header file <*xview/file_chsr.h*>. The object type is `File_chooser`. The class hierarchy for the `FILE_CHOOSER` package is:

```
[Generic] -> [(Drawable)] -> [Window] -> [Frame_cmd] -> [File Chooser]
```

File choosers come in three types:

- `FILE_CHOOSER_OPEN`
- `FILE_CHOOSER_SAVE`
- `FILE_CHOOSER_SAVEAS`

Figure D-1 shows an example of an OPEN file chooser.

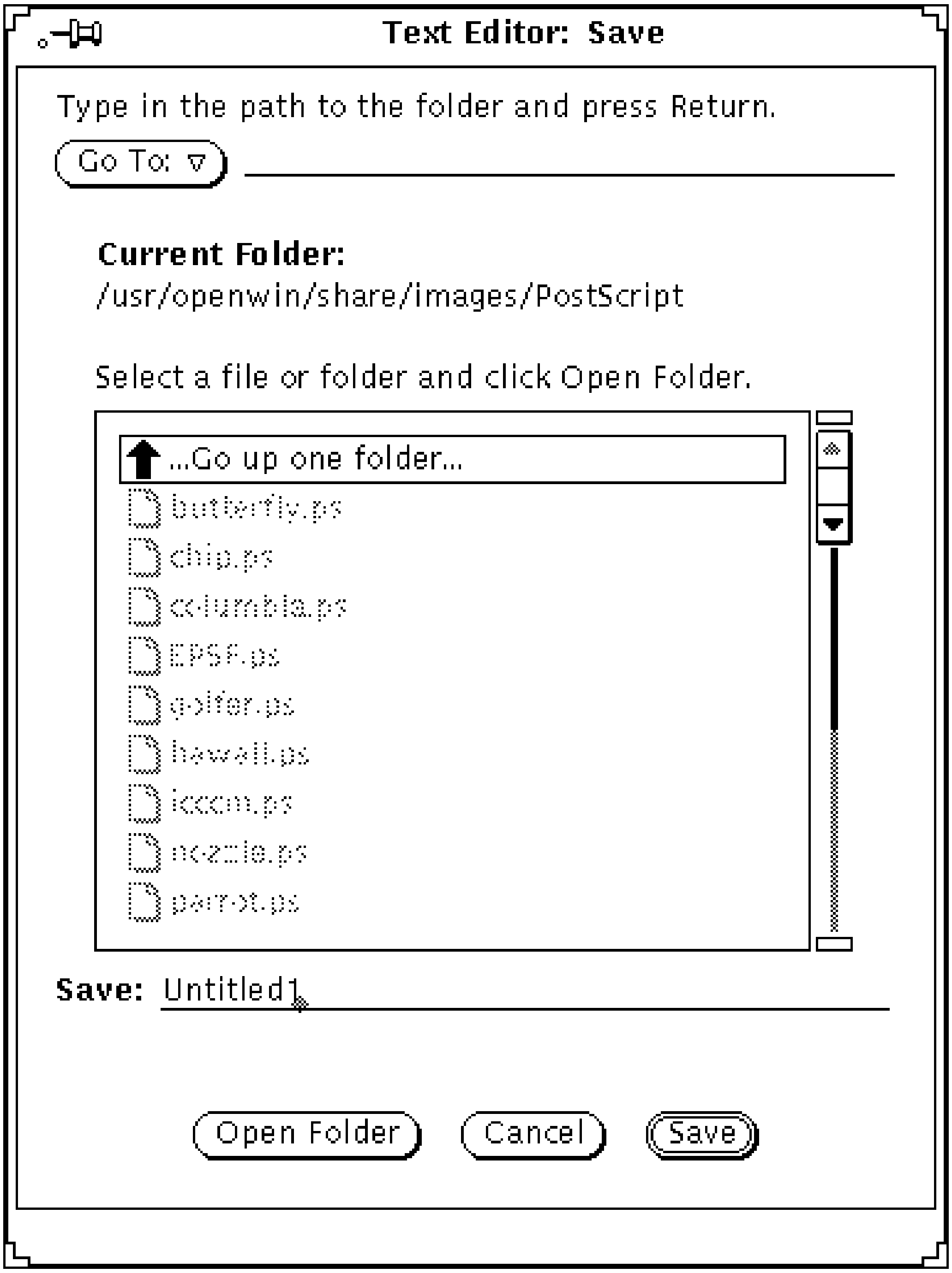

Figure D-1. File chooser Save dialog

Figure D-2 shows an example of a SAVE file chooser which is similar to a SAVE AS file chooser.

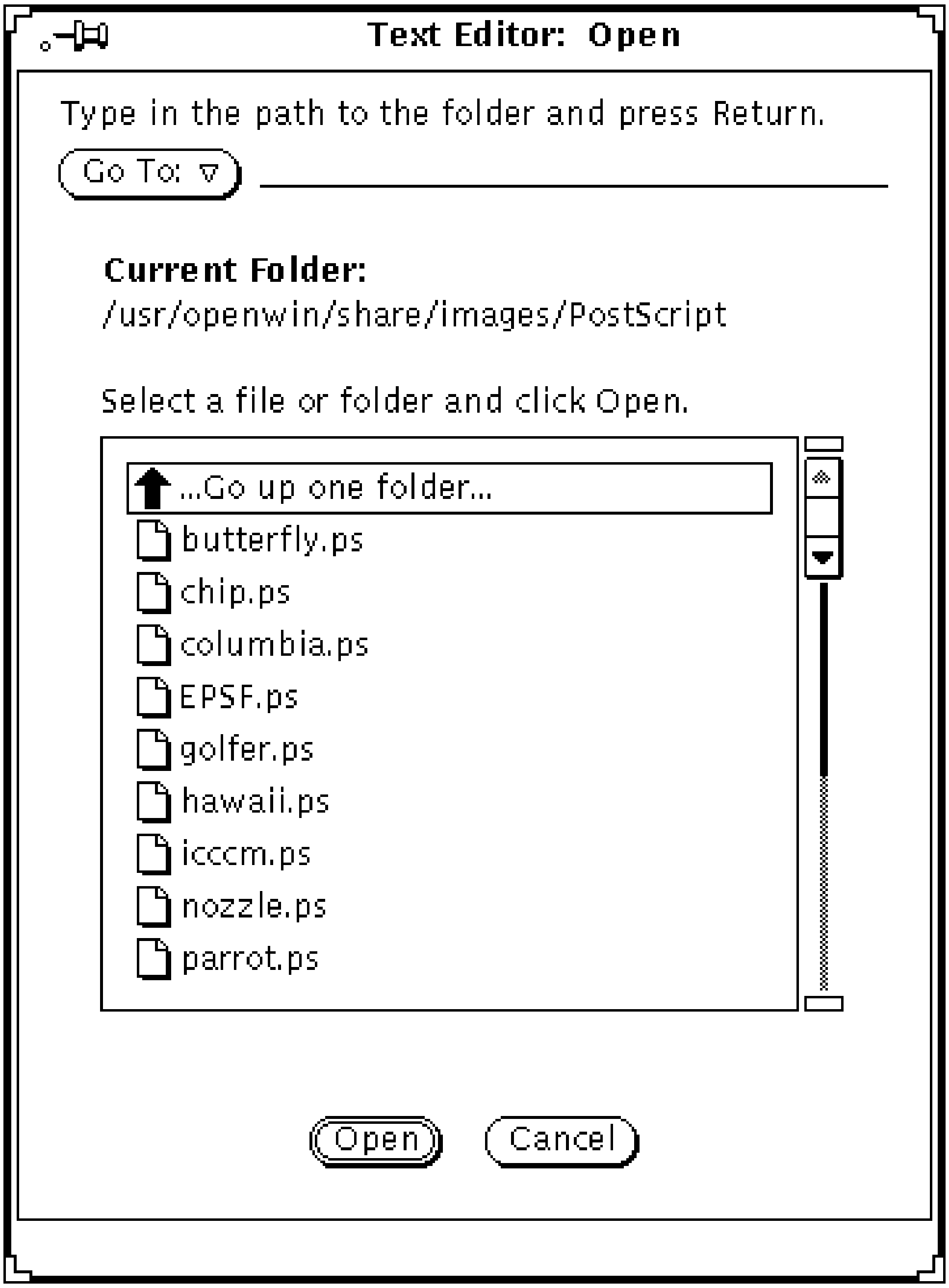

Figure D-2. File chooser Open dialog

These dialogs are presented from an application's file pop-up menu. The user may select one of the predefined package choices: "Open", "Save", "Save As", or an application defined choice such as "Import" or "Include". Once the file chooser is open, the user is presented with a frame containing several elements for navigating the folder hierarchy. These elements include the Go To menu, the Current Folder field, the scrolling list, and the Open, Save or Save As button area, which also includes a Cancel button. The Save and Save As file chooser also include a type in field below the scrolling list where the saved file is named. Refer to the figures for the relative layouts of these file chooser elements. Note: the packages that make up these elements are public and include: `FILE_LIST`, `PATH_NAME`, `HISTORY_LIST` and `HISTORY_MENU`. You can uses these packages for applications that require functionality similar to that provided by any of these file chooser components. Refer to Section A.13, "File Chooser Components" for information on using these packages. File chooser types are specified at create time using the `FILE_CHOOSER_TYPE` attribute. Alternatively, convenience macros allow you to create each of the specified file chooser types as shown below:

```
File_chooser  open_chsr;
File_chooser  save_chsr;
File_chooser  saveas_chsr;

open_chsr   = (File_chooser) xv_create(owner,
                              FILE_CHOOSER_OPEN_DIALOG, NULL);
save_chsr   = (File_chooser) xv_create(owner,
                              FILE_CHOOSER_SAVE_DIALOG, NULL);
saveas_chsr = (File_chooser) xv_create(owner,
                              FILE_CHOOSER_SAVEAS_DIALOG, NULL);
```

The owner of a `File_chooser` should be a base frame, but it can also be the root window, `NULL`, just as with a command frame.

D.2 Using a File Chooser

Given the information provided so far, we can demonstrate to how to install a file chooser for a menu item. Example D-1 shows a portion of a program which installs a File chooser on a menu item. The complete program is found in Appendix F, *Example Programs.*

Example D-1. Portion of file_chooser.c program

```
/*
 * Demonstrate the XView File Chooser
 */
#include <stdio.h>
#include <xview/xview.h>
#include <xview/panel.h>
#include <xview/textsw.h>
#include <xview/scrollbar.h>
#include <xview/file_chsr.h>

static Attr_attribute  MY_KEY;

typedef struct {
    Frame    frame;
```

Example D-1. Portion of file_chooser.c program (continued)

```
    Panel    panel;
    Panel_button_item  file_button;
    Menu    file_menu;
    Textsw    textsw;
    File_chooser  open;
    File_chooser  save;
    File_chooser  saveas;
    File_chooser  import;
    char *    doc_name;
} My_ui;

static void  my_open_notify();
static void  my_save_notify();
static void  my_saveas_notify();
static void  my_import_notify();
static int  my_open_callback();
static int  my_save_callback();

void
main( argc, argv )
     int argc;
     char **argv;
{
    My_ui ui = {0};


    (void) xv_init ( XV_INIT_ARGC_PTR_ARGV,
                       &argc, argv, NULL );
    MY_KEY = xv_unique_key();

    ui.file_menu
  = xv_create(XV_NULL, MENU,
        MENU_ITEM,
          MENU_STRING,    "Open...",
          MENU_NOTIFY_PROC,  my_open_notify,
          NULL,
        MENU_ITEM,
          MENU_STRING,    "Import...",
          MENU_NOTIFY_PROC,  my_import_notify,
          NULL,
        MENU_ITEM,
          MENU_STRING,    "Save...",
          MENU_NOTIFY_PROC,  my_save_notify,
          NULL,
        MENU_ITEM,
          MENU_STRING,    "Save As...",
          MENU_NOTIFY_PROC,  my_saveas_notify,
          NULL,
        XV_KEY_DATA,    MY_KEY, &ui,
        NULL);

    ui.frame = xv_create(XV_NULL, FRAME,
       XV_LABEL,    "Demo Text Editor",
       FRAME_SHOW_FOOTER,  TRUE,
       NULL );
```

Example D-1. Portion of file_chooser.c program (continued)

```
    ui.panel = xv_create(ui.frame, PANEL, NULL);

    ui.file_button = xv_create(ui.panel, PANEL_BUTTON,
             PANEL_LABEL_STRING,  "File",
             PANEL_ITEM_MENU,    ui.file_menu,
             NULL);

    window_fit_height( ui.panel );

    ui.textsw = xv_create(ui.frame, TEXTSW,
        XV_X,    0,
        WIN_BELOW,  ui.panel,
        NULL);

    xv_main_loop( ui.frame );
    exit( 0 );
}

/*
 * Picked "Open" off of File Menu.
 */
static void
my_open_notify( menu,  mi )
     Menu menu;
     Menu_item mi;
{
    My_ui *ui = (My_ui *)xv_get(menu, XV_KEY_DATA, MY_KEY);

    if ( !ui->open ) {
  ui->open
      = xv_create(ui->frame, FILE_CHOOSER_OPEN_DIALOG,
      XV_LABEL,      "Text Editor:  Open",
      FILE_CHOOSER_NOTIFY_FUNC,  my_open_callback,
      XV_KEY_DATA,      MY_KEY, ui,
      NULL);
    }

    xv_set(ui->open, XV_SHOW, TRUE, NULL);
}

/*
 * Picked OPEN off of File Menu.
 *  See Appendix F for full example
 */

my_open_callback( fc, path, file, client_data )
     File_chooser fc;
     char *path;
     char *file;
     Xv_opaque client_data;
{
    My_ui *ui = (My_ui *)xv_get(fc, XV_KEY_DATA, MY_KEY);
    Textsw_status status;
    char buf[512];

    xv_set(fc, FRAME_BUSY, TRUE, NULL);
```

Example D-1. Portion of file_chooser.c program (continued)

```
    xv_set(ui->textsw,
     TEXTSW_STATUS, &status,
     TEXTSW_FILE,  path,
     TEXTSW_FIRST, 0,
     NULL);

    if ( status != TEXTSW_STATUS_OKAY ) {
  window_bell( ui->frame );
  xv_set( ui->frame,
         FRAME_LEFT_FOOTER, "Unable to load file!",
         NULL);
  xv_set(fc, FRAME_BUSY, FALSE, NULL);
  return XV_ERROR;
    }

    /* Set current doc name on the Save popup. */
    (void) sprintf(buf, "%s.1", file);
    if ( ui->saveas )
  xv_set(ui->saveas, FILE_CHOOSER_DOC_NAME, buf, NULL);
    else {
  if ( ui->doc_name )
      free( ui->doc_name );
  ui->doc_name = strdup( buf );
    }

    (void) sprintf(buf, "Demo Text Editor – %s", file);
    xv_set(ui->frame,
     XV_LABEL,  buf,
     NULL);

    xv_set(fc, FRAME_BUSY, FALSE, NULL);

    return XV_OK;
}
.
.
.
```

D.3 Notification from a File Chooser

When a user presses the "Open" or "Save" button on a File chooser, the `FILE_CHOOSER` package first tries to validate the given path name. If the path name passes the validation, the client is notified using the notify function specified with `FILE_CHOOSER_NOTIFY_FUNC`. If the attribute `FILE_CHOOSER_NO_CONFIRM` is set to `TRUE`, the validation step does not occur.

The callbacks for the Open and Save operations are different. The Open Callback takes the form:

```
int
open_callback( fc, path, file, client_data )
  File_chooser  fc;
```

Version 3.2 and the File Chooser

```
    char *        path;
    char *        file;
    Xv_opaque     client_data;
```

The `path` argument is the full path to the file. Use the attribute `FILE_CHOOSER_DIRECTORY` to obtain the path. The `file` argument is strictly the file-name portion of the path. The `client_data` arugment is the `client_data` field set for the row in the display list from the `FILE_CHOOSER_FILTER_FUNC`.

The Save and Save As callbacks take the form:

```
int
save_callback( fc, path, stats )
   File_chooser   fc;
   char *         path;
   struct stat *  stats;
```

The `path` parameter is the full path of the file to be saved. The `stats` parameter is a pointer to the file's `stat` structure, if the file exists, or `NULL` if the file does not exist.

D.4 Controlling the File Chooser Display List

The view of the file system that the `FILE_CHOOSER` displays in the scrolling list (the file list) can be controlled by the client using a system of three callbacks and/or a regular expression string. Using these file list controls, the file chooser can gray out files that are not selectable. An application may choose to gray out object files (.o files) or files otherwise not openable for the particular application.

The flow of control for loading a new directory into the file chooser's file list is as follows:

1. Call the callback specified with the `FILE_CHOOSER_CD_FUNC` attribute with an `op` of `FILE_CHOOSER_BEFORE_CD`. If `XV_ERROR` is returned, leave the list empty and return.
2. Match each file against the regular expression given with the `FILE_CHOOSER_FILTER_STRING` attribute This attribute is used primarily with the OPEN type file chooser to gray out files in the file list that cannot be opened.
3. If the mask specified with `FILE_CHOOSER_FILTER_MASK` matches the file attributes, then invoke the `FILE_CHOOSER_FILTER_FUNC`. If the callback *filter-func* returns `FILE_CHOOSER_IGNORE`, gray out the file name.
4. After reading all the files, sort them using the order specified by the callback installed with `FILE_CHOOSER_COMPARE_FUNC`.
5. After sorting, insert the list of files into the `FILE_LIST` object without repainting and call the `FILE_CHOOSER_CD_FUNC` with an `op` of `FILE_CHOOSER_AFTER_CD`.
6. Finally, redisplay the list of files as the new contents of File chooser. This also adds the current folder into the Goto Menu.

These steps, and control sequence for the attributes in the file chooser are summarized in the following control flow diagram. More details are presented in the following sections.

```
                                   /* FILE_CHOOSER_CD_FUNC */
if ( call cd_func(before-cd) != XV_OK )
   return

while ( more files ) {
                                   /* FILE_CHOOSER_FILTER_STRING */
    matched = current file / filter-string
                                   /* FILE_CHOOSER_MATCH_GLYPH */
    assign default glyph
                                   /* FILE_CHOOSER_FILTER_MASK */
                                   /* FILE_CHOOSER_FILTER_FUNC */
    if ( current file in filter-mask )
      call filter-func

    if ( file accepted )           /* FILE_CHOOSER_ABREV_VIEW */
      add to list
}
                                   /* FILE_CHOOSER_COMPARE_FUNC */
sort the list, using the compare_func

insert the new list of files into Scrolling List
                                   /* FILE_CHOOSER_CD_FUNC */
call cd_func(after-cd)

redisplay Scrolling List to user
```

D.4.1 Monitoring Directory Changes

An application may require the flexibility to intercept a change of directory or to modify the list of files after the file list has been built and before it is shown to the user. Setting the `FILE_CHOOSER_CD_FUNC` attribute installs a callback that is issued twice during a directory change. The first time, it is called with a `File_chooser_op` of `FILE_CHOOSER_BEFORE_CD`, this allows the client to return `XV_ERROR` to prevent the directory change. The change directory function could be called at this stage to veto a change of directory if the application treats a directory specially. For example, Interleaf uses a directory to store book files, that make up a single “document”. Thus a Interleaf application could open a directory as a document rather than the individual files in the directory.

The second invocation of the change directory function callback is after the dispay list has been built. The callback’s `op` parameter is set to `FILE_CHOOSER_AFTER_CD`. This allows the client to modify the file list before the user sees it. If `xv_set` calls are used on the file list from inside the change directory callback, be sure to set `PANEL_PAINT` for the file list to `PANEL_NONE`.

D.4.2 Filtering

Files which are not “openable” by the application will be grayed out in the disply list. The application may select files for special filtering using FILE_CHOOSER_FILTER_STRING and/or FILE_CHOOSER_FILTER_MASK.

Using FILE_CHOOSER_FILTER_STRING, an application may specify a regular expression to match parts of a file name such as a common postfix (for example .c for C source files). Files that match this expression will then be grayed out in the file list. Along with this regular expression matching, the application may specify a Server_image that automatically is displayed for each file that matches. The Server_image for matched files is specified by FILE_CHOOSER_MATCH_GLYPH and FILE_CHOOSER_MATCH_GLYPH_MASK. By default, the filter string is set to NULL which disables pattern matching entirely.

Another method for filtering the file list is by specifing a callback that allows the application to qualify each file and return information such as a Server_image to be displayed with the file and a client_data field to return to the “open” notify callback. The filter callback is installed using FILE_CHOOSER_FILTER_FUNC. The files for wich this callback gets issued can be selected with the FILE_CHOOSER_FILTER_MASK attribute which takes an or’d list of type File_chooser_filter_mask flags.

D.4.3 File Chooser Sorting

The default sorting algorithm used by the FILE_CHOOSER package is alpha-numeric without case sensitivity. If a different sort algorithm is appropriate for a particular application, the different sort routine may be installed with the FILE_CHOOSER_COMPARE_FUNC attribute. This attribute works using the qsort(3) library routine.

XView provides several built-in comparison functions for ascending and descending case-sensitive or case-insensitive sorts. Also, the comparison functions are given arguments of type File_chooser_row which includes information such as the file name, the associated stat(3) structure, the strxfrm(3) version of the string and the File_chooser_op of FILE_CHOOSER_MATCHED or FILE_CHOOSER_NOT_MATCHED with respect to the FILE_CHOOSER_FILTER_STRING. This provides a highly flexible and efficient means of sorting files.

D.5 Modifying the Display List

There are several attributes that allow the client to modify the display of files in a file chooser. A file chooser's default values comply with the OPEN LOOK Application File Choosing Specification. Note that the File Choosing Specification requires the client to provide an appropriate human interface to revert to the default behavior if the client changes the default behavior (for example, the application should provide a toggle that turns off the display of dot files if the application displays dot files.)

D.5.1 Dot Files

Since most dot files are created by applications and not directly by the user, the file chooser does not show dot files by default (also, user testing has shown that dot files often confuse end-users). If an application needs to show dot files in the file list display, the attribute `FILE_CHOOSER_SHOW_DOT_FILES` toggles their display.

D.5.2 Abbreviated View

Some applications may only need to show those files that are relevant to the application. In this case, files that are not openable may be removed from the file list, rather than being displayed as grayed out items. This functionality may be turned on by setting the `FILE_CHOOSER_ABBREV_VIEW` attribute to `TRUE`. Warning: use of this attribute is recommended only for applications that expect very technical users who will understand that files not being displayed is not a problem.

D.6 File Chooser Customization

The application has the option of adding permanent application-specific entries into the file chooser Go To menu. The attribute `FILE_CHOOSER_APP_DIR` sets the values for these entries.

Some applications may need to do use a non-standard configuration with the Go To menu. This is achieved by creating a custom `History_list` object and attaching it to a file chooser with the `FILE_CHOOSER_HISTORY_LIST` attribute. Once installed, the `FILE_CHOOSER` package will add "Recent Entries" to the installed list as the user navigates the file system.

Other modifications for the Go To history list and menu can be accomplished using the `HISTORY_LIST` and `HISTORY_MENU` attributes. The last section of this chapter list the `HISTORY` attributes. Refer to the XView Reference Manual for additional information on these attributes.

The Save and Save As FILE_CHOOSERs can include a default name in the field for the File type in. You specify this default name using the FILE_CHOOSER_DOC_NAME attribute. For a Save file chooser, the default document name should be Untitled1. For a Save As File Chooser, the default document name should be "*current document*".1.

In some circumstances, the application may wish to save more than one file to a directory. The file chooser Save dialog does not provide for this very well, so there is an option to gray out the Save typein while leaving the Save button available to the user. This is toggled with the FILE_CHOOSER_SAVE_TO_DIR attribute.

Since the file chooser has elements that are public it is possible to get handles to the objects that make up the file chooser, either by the FILE_CHOOSER_CHILD interface or by PANEL_EACH_ITEM, etc. For example, the file chooser's FILE_LIST object is available in this manner. Keep in mind that values that are not gotten or set via the parent FILE_CHOOSER API are *not* guaranteed to be compatible across releases of the XView toolkit.

The attribute FILE_CHOOSER_AUTO_UPDATE modifies the action of a FILE_CHOOSER. This attribute tells the FILE_CHOOSER to re-read its current directory only when FILE_CHOOSER_UPDATE is explicitly called, or when the user performs some action to read the directory.

D.7 Customizing the File Chooser Dialog

The OPEN LOOK Application File Choosing Specification defines two methods for OPEN LOOK-Compliant file chooser customization. The first, and simplest, is to modify the Open dialog and define a custom File Chooser, such as an "Import" or an "Include" File Chooser. Using the attributes supplied for adding a custom button, this change is not difficult. For example,

```
File_chooser open_chsr;

open_chsr = (File_chooser) xv_create(owner,
                      FILE_CHOOSER_OPEN_DIALOG,
                      FILE_CHOOSER_CUSTOMIZE_OPEN,
                        "Import",
                        "Select a file or folder and click Import",
                      FILE_CHOOSER_SELECT_FILES,
                      NULL);
```

The first argument supplied to the attribute FILE_CHOOSER_CUSTOMIZE_OPEN is the label for the new custom button. The second argument is the line of text that is placed above the Scrolling List. The third argument is an enum value that represents whether this custom operation can deal with files (that is, if the item can only use files this argument is set to FILE_CHOOSER_SELECT_FILES, otherwise, if both files and directories are valid it is set to FILE_CHOOSER_SELECT_ALL).

The second way to customize the XView file chooser is to add custom controls to the file chooser panel. Implementing this is a multi-step process.

The OPEN LOOK Application File Choosing Specification mandates that custom controls be placed under the Scrolling List (and below the Save typein), but above the buttons at the bottom of the dialog. To implement this, the `FILE_CHOOSER` package reserves an area, called the *extension rectangle*, in part of the dialog space. By default, the extension rectangle has a height of 0 pixels; the client make may this area a specific size using the `FILE_CHOOSER_EXTEN_HEIGHT` attribute.

The `FILE_CHOOSER` package leaves much of the responsibility for layout and sizing of custom controls in the hands of the client. The client has the responsibility of positioning custom controls within the extention rectangle area during layout and resizing. The callback installed with the attribute `FILE_CHOOSER_EXTEN_FUNC` handles resizing and layout.

Other responsibilities of the client when adding custom controls include adjusting the default and minimum sizes for the Frame.

Example D-2 illustrates adding an extension item to a file chooser.

Example D-2. An extension item program

```
File_chooser fc;
Panel        panel;
Panel_item   item;
int          item_width;
int          item_height;
int          frame_width;
int          frame_height;

panel = xv_get(fc, FRAME_CMD_PANEL);

item = xv_create(panel, PANEL_CHOICE,
         PANEL_LABEL_STRING,    "Hidden Files:",
         PANEL_CHOICE_STRINGS,  "Hide", "Show", NULL,
         PANEL_NOTIFY_PROC,      my_show_dot_files_proc,
         NULL);

item_width = (int) xv_get(item, XV_WIDTH);
item_height = (int) xv_get(item, XV_HEIGHT);


/*
* Adjust Frame default size to make room for the extension item.
*/
frame_width = (int) xv_get(fc, XV_WIDTH);
frame_height = (int) xv_get(fc, XV_HEIGHT);
xv_set(fc,
     XV_WIDTH,  MAX(frame_width, (item_width + xv_cols(panel, 4))),
     XV_HEIGHT, frame_height + item_height,
     NULL);


/*
* Adjust Frame Min Size.  provide for at least 2
* columns on either side of the extension item.
*/
xv_get(fc, FRAME_MIN_SIZE, &frame_width, &frame_height);
xv_set(fc,
```

Example D-2. An extension item program (continued)

```
    FRAME_MIN_SIZE, MAX( frame_width, (item_width + xv_cols(panel, 4))),
                    frame_height + item_height,
    NULL);


/* Tell file chooser to reserve layout space for it */

xv_set(fc,
     FILE_CHOOSER_EXTEN_HEIGHT,  item_height,
     FILE_CHOOSER_EXTEN_FUNC,    my_exten_func,
     XV_KEY_DATA, EXTEN_ITEM_KEY, item,
     NULL);


/*   [.................]  */


static int
my_exten_func( fc, frame_rect, exten_rect,
               left_edge, right_edge, max_height )
     File_chooser fc;
     Rect *frame_rect;
     Rect *exten_rect;
     int left_edge;
     int right_edge;
     int max_height;
{
    Panel_item item = (Panel_item) xv_get(fc,
                                    XV_KEY_DATA, EXTEN_ITEM_KEY);
    int item_width;

    item_width = (int) xv_get(item, XV_WIDTH);

/*
* show item centered in frame.
*/
    xv_set(item,
     XV_X,  (frame_rect->r_width - item_width) / 2,
     XV_Y,  exten_rect->r_top,
     PANEL_PAINT, PANEL_NONE,
     NULL);

    return -1;    /* (-1) means exten height didn't change */
}
```

D.7.1 File Chooser Components

Internally, the file chooser is uses three high-level interfaces: the `FILE_LIST` package, the `PATH_NAME` packge, and the `HISTORY` package. The `FILE_LIST` package is a subclass of `PANEL_LIST`; it handles all of the navigation and display of the UNIX File System. The `PATH_NAME` package is a subclass of `PANEL_TEXT`; it handles shell variable and tilde expansion. The `HISTORY` package implements a shareable command history through the XView

Menu and Menu_item packages (the Go To menus in the file chooser use this). If your file manipulation needs do not fall within the scope of the OPEN LOOK Application File Choosing Specification, you have the option of designing your own dialog with the same components the File uses. Since the FILE_CHOOSER package implements a particular look and feel, which may not suffice for all application needs, the component parts are made available.

```
[Generic] -> [Panel_item] -> [Panel_list_item] -> [File_list]
[Generic] -> [Panel_item] -> [Panel_text_item] -> [Path_name]
[Generic] -> [History_menu]
[Generic] -> [History_list]
```

D.8 Version 3.2 Additions

This section lists additional changes for available with XView Version 3.2 and new releases.

D.8.1 New Panel List Attributes for Version 3.2

Version 3.2 of XView offers a new and improved method for adding entries to a PANEL_LIST. Version 3.2 offers the following new attributes for panel lists:

```
PANEL_LIST_INACTIVE
PANEL_LIST_DELETE_INACTIVE_ROWS
PANEL_LIST_DO_DBL_CLICK
PANEL_LIST_MASK_GLYPH
PANEL_LIST_MASK_GLYPHS
PANEL_LIST_ROW_VALUES
PANEL_LIST_EXTENSION_DATA
PANEL_LIST_EXTENSION_DATAS
```

D.8.1.1 Adding new list entries

The attribute PANEL_LIST_ROW_VALUES offers an improved performance method of getting/setting row values in a PANEL_LIST. This attribute takes the row number, a pointer to a Panel_list_row_values array, and a count of how many rows in the array. The enum Panel_list_row_values is defined as shown below:

```
typedef struct {
char * string;
Server_image glyph;
Server_image mask_glyph;
Xv_font font;
Xv_opaque client_data;
Xv_opaque extension_data;
unsigned inactive : 1;
unsigned selected : 1;
} Panel_list_row_values;
```

Version 3.2 and the File Chooser

On `get`, the arguments remain the same, the array passed in gets filled in by the `PANEL_LIST` package. The return value is the number of rows that were successfully filled in.

Example D-3 shows a portion of a program which uses the new `PANEL_LIST` insertion method.

Example D-3. Program that adds values to a panel list

```
/*
 * Demonstrate the use of the PANEL_LIST_ROW_VALUES attribute
 */

#include <stdio.h>
#include <xview/xview.h>
#include <xview/font.h>
#include <xview/panel.h>


static Attr_attribute MY_KEY;

static void  my_clear_proc();
static void  my_load_proc();
static void  my_print_proc();

typedef struct {
    Frame frame;
    Panel_list_item list;
    Xv_font font;
} My_ui;


void
main ( argc, argv )
     int argc;
     char **argv;
{
    Panel panel;
    My_ui ui;

    (void) xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, NULL);
    MY_KEY = xv_unique_key();

    ui.frame = xv_create ( XV_NULL, FRAME,
        XV_LABEL,    "New Load",
        FRAME_SHOW_FOOTER,  TRUE,
        NULL );
    panel = xv_create ( ui.frame, PANEL, NULL );


    (void) xv_create ( panel, PANEL_BUTTON,
          PANEL_LABEL_STRING,    "Load",
          PANEL_NOTIFY_PROC,    my_load_proc,
          XV_KEY_DATA,      MY_KEY, &ui,
          NULL );

    (void) xv_create ( panel, PANEL_BUTTON,
          PANEL_LABEL_STRING,    "Clear",
```

Example D-3. Program that adds values to a panel list (continued)

```
          PANEL_NOTIFY_PROC,    my_clear_proc,
          XV_KEY_DATA,      MY_KEY, &ui,
          NULL );

    (void) xv_create( panel, PANEL_BUTTON,
          PANEL_LABEL_STRING,    "Print Selected Row",
          PANEL_NOTIFY_PROC,    my_print_proc,
          XV_KEY_DATA,      MY_KEY, &ui,
          NULL );

    ui.list = xv_create( panel, PANEL_LIST,
      PANEL_LIST_WIDTH,    300,
      PANEL_LIST_DISPLAY_ROWS,  8,
      NULL);

    ui.font = xv_create(XV_NULL, FONT,
      FONT_FAMILY,  FONT_FAMILY_DEFAULT,
      FONT_STYLE,  FONT_STYLE_BOLD,
      NULL);

    window_fit ( panel );
    window_fit ( ui.frame );
    xv_main_loop ( ui.frame );
}


static void
my_clear_proc( item, event )
     Panel_item item;
     Event *event;
{
    My_ui *ui = (My_ui *)xv_get(item, XV_KEY_DATA, MY_KEY);
    int rows = (int)xv_get(ui->list, PANEL_LIST_NROWS);

    if ( rows > 0 )
  xv_set(ui->list,
         PANEL_LIST_DELETE_ROWS,  0, rows,
         NULL);
    xv_set(ui->frame, FRAME_LEFT_FOOTER, "", NULL);
}


static void
my_load_proc( item, event )
     Panel_item item;
     Event *event;
{
    My_ui *ui = (My_ui *)xv_get(item, XV_KEY_DATA, MY_KEY);
    int ii;
    Panel_list_row_values vals[1000];
    char buf[64];

    xv_set(ui->frame, FRAME_BUSY, TRUE, NULL);

    for(ii=0; ii<1000; ++ii) {
  vals[ii].string = "Testing PANEL_LIST_ROW_VALUES";
```

Example D-3. Program that adds values to a panel list (continued)

```
  vals[ii].font = ui->font;
  vals[ii].glyph = XV_NULL;
  vals[ii].mask_glyph = XV_NULL;
  vals[ii].client_data = ui->font;
  vals[ii].selected = FALSE;
  vals[ii].inactive = FALSE;
    }

    xv_set(ui->list,
     PANEL_LIST_ROW_VALUES,  0, &vals, 1000,
     NULL);

    (void) sprintf(buf, "%d rows in list",
       (int) xv_get(ui->list, PANEL_LIST_NROWS)
       );
    xv_set(ui->frame,
     FRAME_LEFT_FOOTER,  buf,
     FRAME_BUSY, FALSE,
     NULL);
}



static void
my_print_proc( item, event )
     Panel_item item;
     Event *event;
{
    My_ui *ui = (My_ui *)xv_get(item, XV_KEY_DATA, MY_KEY);
    Panel_list_row_values vals;
    int row = (int) xv_get(ui->list, PANEL_LIST_FIRST_SELECTED);
    int count;

    count = (int) xv_get(ui->list, PANEL_LIST_ROW_VALUES, row, &vals, 1);

    if ( count != 1 ) {
  window_bell( ui->frame );
  xv_set(ui->frame,
         FRAME_LEFT_FOOTER,  "Unable to get row",
         FRAME_BUSY,     FALSE,
         NULL);
  return;
    }

    printf( "Row Number %d:0, row );
    printf( "   String: %s0, vals.string );
    printf( "   Selected: %d0, vals.selected );
    printf( "   Inactive: %d0, vals.inactive );
}
```

D.8.1.2 Other panel list changes

This section describes the additional changes for the panel list package.

The attribute PANEL_LIST_INACTIVE "Grays out" a row in a PANEL_LIST. Note that a row that is inactive cannot be selected. Also, mouseless model navigation is not effected by the inactive state of the individual rows.

The attribute PANEL_LIST_DELETE_INACTIVE_ROWS deletes all inactive rows from the list. This is similar to PANEL_LIST_DELETE_SELECTED_ROWS attribute.

The PANEL_LIST_DO_DBL_CLICK attribute tells PANEL_LIST to interpret two select events that occur within the timout value as a double-click instead of as a second select or a deselect (depending on the current mode of the list). The timeout value is specified with OpenWindows.MulticlickTimeout If true, the PANEL_LIST will deliver a new op called PANEL_LIST_OP_DBL_CLICK instead of the normal PANEL_LIST_OP_SELECT or PANEL_LIST_OP_DESELECT.

The attributes PANEL_LIST_MASK_GLYPH and PANEL_LIST_MASK_GLYPHS tell the PANEL_LIST to use the given Server_image as a clip mask for the corresponding PANEL_LIST_GLYPH. The Server_image supplied must be of depth 1. PANEL_LIST_MASK_GLYPHS is like like PANEL_LIST_MASK_GLYPH, but takes a NULL terminated list of glyphs rather than a row and a single handle.

The attributes PANEL_LIST_EXTENSION_DATA and PANEL_LIST_EXTENSION_DATAS are the same as PANEL_LIST_CLIENT_DATA, except they are reserved for package implementors. This is used by the FILE_LIST package.

D.9 Keyboard Menu Accelerators

The keyboard menu accelerator functionality in XView provides attributes to associate an accelerator with a menu item. Keyboard menu accelerators can be used to invoke menu commands directly without having to display the menu in which the commands appear. These accelerators provide a more efficient path to familiar menu functionality.

Accelerators are global to a frame of an application. The accelerator key strokes are displayed on the right side of the menu item. The diamond symbol represents the meta key. If there are qualifiers such as Control (ctrl), Shift (shift) or Alt (alt), indicated on the menu item, those keys are to be used in conjunction with the accelerator key and possibly the Meta key (if the diamond symbol exists on the menu item).

If a menu is pinned, the accelerators will not be displayed on the menu, as a way to conserve screen space. Although the accelerators are not visible on the menu, they are still active and can be used as long as the input focus is in the application frame where the menu was first brought up.

Menu accelerators are activated if the resource OpenWindows.MenuAccelerators is set to True (the default case for OpenWindows Version 3.2). This resource can be set using the "Keyboard" category of the OpenWindows(Version 3.2) Workspace Properties program. Setting "Keyboard Menu Equivalents" to "Application + Window" or "Application Only" sets the OpenWindows.MenuAccelerators resource to True, while the "None" setting will set the resource to False.

D.9.0.1 Frame package menu accelerator attributes

The attributes FRAME_MENUS, FRAME_MENU_COUNT, FRAME_MENU_ADD, and FRAME_MENU_DELETE are used in conjunction with menu accelerators. They are used to inform the frame object which menus will be used on the frame - this is required because the frame has to know what menu accelerators to detect. Any menus with accelerators that are used anywhere on the frame, i.e. on panels, textsw, canvas will not work until they are registered using these attributes.

The attribute FRAME_MENUS replaces the current menu list with the one passed on the avlist. For get, this return a pointer to the current list of menus. The list returned should not be modified by the application.

```
Menu   *menu_list;

xv_set(frame1, FRAME_MENUS,
                edit_menu, load_menu, NULL,
               NULL);

menu_list = (Menu *)xv_get(frame1, FRAME_MENUS);
```

The number of menus can be obtained with FRAME_MENU_COUNT.

```
int   menu_count;

menu_count = (int)xv_get(frame2, FRAME_MENU_COUNT);
```

The attribute FRAME_MENU_COUNT returns the current number of menus registered on the frame via FRAME_MENUS, FRAME_MENU_ADD, or FRAME_MENU_DELETE.

The attribute FRAME_MENU_ADD appends to the list of accelerated menus on the frame. For example:

```
xv_set(frame1,
     FRAME_MENU_ADD, print_menu,
     NULL);
```

The attribute FRAME_MENU_DELETE deletes from the list of accelerated menus on the frame.

```
xv_set(frame1,
     FRAME_MENU_DELETE, print_menu,
     NULL);
```

D.9.0.2 The menu attributes for menu accelerators

The following attributes provide functionality to define menu accelerators for menu items. The basic difference for menu accelerators is that when an accelerator (eg. Meta+l) is pressed, the menu is not brought up. Instead, the procedures that are normally called when one selects a menu item, are called directly.

Each new attribute below corresponds to one of many ways of creating a menu item. The attribute `MENU_STRINGS_AND_ACCELERATORS` and `MENU_ACTION_ACCELERATOR` are for static creation of menu items for a given menu.

The attribute `MENU_ACCELERATOR`, when used in a create or set call, set an accelerator on a menu item:

```
Menu   menu;

menu = xv_create(NULL, MENU
             MENU_ITEM,
                  MENU_STRING, "Load",
                  MENU_NOTIFY_PROC, load_proc,
                  MENU_ACCELERATOR, "Meta+l",
             NULL,
             NULL);
```

or

```
Menu_item  load_item;

load_item = xv_create(NULL, MENU_ITEM,
                    MENU_STRING, "Load",
                    MENU_NOTIFY_PROC, load_proc,
                    MENU_ACCELERATOR, "Meta+l",
                    NULL);
```

The accelerator string will be copied by XView. A get returns the accelerator string. The string returned should not be modified.

The attribute `MENU_ACTION_ACCELERATOR` can be used to create a menu item with a given label, notify procedure, and accelerator:

```
xv_set(menu,
         MENU_ACTION_ACCELERATOR,
             "Load", load_proc, "Meta+L",
         NULL);
```

The menu item label string, the first argument, will not be copied by XView. The accelerator string, argument three, will be copied.

`MENU_STRINGS_AND_ACCELERATORS` can be used to create a menu item with a given label and accelerator:

```
xv_set(menu, MENU_NOTIFY_PROC, file_proc,
             MENU_STRINGS_AND_ACCELERATORS,
                "Load", "Meta+l",
                "Print", "Meta+p",
                "Include","Meta+Ctrl+i",
             NULL,
             NULL);
```

The accelerator strings will be copied by XView. The menu item label strings will not be copied.

For all the attributes above, if an accelerator is *changed* with `xv_set`, the attribute `FRAME_MENUS` must be set again before the change to the accelerator will take effect.

Callback procedures registered using `MENU_NOTIFY_PROC`, `MENU_GEN_PROC`, and `MENU_DONE_PROC` will be called the same way for menu accelerators as if the menu item was selected using the menu directly.

The accelerators can be specified in a number of ways. These specifications are a combination of Xt, OLIT, and XView syntaxes:

```
Xt:             [modifier...] '<Key>'key
OLIT:           [OLITmodifier...] '<'key'>'
XView:          [modifier ['+' modifier] '+'] key
modifier:       'Meta' | 'Shift' | 'Alt' | 'Hyper' | 'Ctrl' |
OLITmodifier:   modifier | 'm' | 's' | 'a' | 'h' | 'c'
```

Key: all print characters and X keysym names (e.g. 'return', 'tab', 'comma', 'period', etc . . .)

Note: keysym names consist of the entries in <X11/keysymdef.h> without the “XK_” prefix. For example:

```
Meta+Shift+a
Meta+comma
Meta <Key>c
Shift Meta <Key>I
m <z>
```

D.9.0.3 Resources

The resource <appname>.<menu item instance name>.accelerator can be used to override the accelerator specified using the attributes described above.

For example, in the application 'foo', if we have:

```
load_item = xv_create(NULL, MENUITEM,
                           XV_INSTANCE_NAME, "load",
                           MENU_STRING, "load",
                           MENU_ACCELERATOR, "Meta+l",
                           MENU_NOTIFY_PROC, load_action,
                      NULL);
```

The "Meta+l” can be overriden by having the following entry in your X resource database:

```
foo.load.accelerator:Meta+b
```

The load action will be done when “Meta+b” is pressed instead of “Meta+l”. Note: `XV_INSTANCE_NAME` will need to be used to give the menu item an instance name.

D.9.0.4 Core set menu accelerators

Various menu accelerators for “common features” of OpenWindows (e.g. Print, Save, Quit, Undo, Paste, etc.) are called the core set menu accelerators. These accelerators are special in that they are specified using the string: "coreset <core set name>" instead of the syntax mentioned above. For example:

```
load_item = xv_create(NULL, MENUITEM,
                      MENU_STRING, "Open",
                      MENU_ACCELERATOR, "coreset Open",
                      MENU_NOTIFY_PROC, load_action,
                      NULL);
```

The core set menu accelerators will be activated only if the X resources:

```
OpenWindows.MenuAccelerator.<core set name>:<accelerator string>
```

are present. For example:

```
OpenWindows.MenuAccelerator.Open:Meta<key>o
```

This means that the key combination meta-o is the menu accelerator for the “open” action. These resources can also be used to rebind the core set accelerators. The advantage of core set accelerators is that all applications that use them will have uniform accelerators (use the same key bindings for the same functions). Rebinding using the resource above will affect all such applications.

The list of possible values of <core set name> and the default case-sensitive resource bindings for OpenWindows.MenuAccelerator.<core set name> are as follows:

Core set accelerator	Default binding
BoldFont	Shift Meta <Key>B
Copy	Meta<Key>c
Cut	Meta <Key>x
Find	Meta <Key>f
ItalicFont	Shift Meta <Key>I
New	Meta <Key>n
NormalFont	Shift Meta <Key>N
Open	Meta <Key>o
Paste	Meta <Key>v
Print	Meta <Key>p
Props	Meta <Key>i
Redo	Shift Meta <Key>Z
Save	Meta <Key>s
SelectAll	Meta <Key>a
Typeface	Meta <Key>t
Undo	Meta <Key>z

In OpenWindows 3.2, the above bindings are set by default upon startup.

D.9.0.5 Events

When a menu is brought up using `menu_show()` and a menu item is selected, it is possible to query the menu using the attribute `MENU_FIRST_EVENT` to obtain the event `(event_action(event) == ACTION_MENU)` that was passed into `menu_show()`, presumably the event that caused the menu to be brought up.

```
Event  *menu_event;

menu_event =  (Event *)xv_get(menu, MENU_FIRST_EVENT);
```

When a menu item notify procedure is called using a menu accelerator, `MENU_FIRST_EVENT` will return the event corresponding to the accelerator. That is, the event that corresponds to "Meta+l". The action id, obtained using the `event_action()` macro, will be `ACTION_ACCELERATOR`. The window id in the event will be the subwindow that had input focus (and received the key event) at the time.

D.10 File Chooser and Version 3.2 Additions Summary

Table D-1 lists the procedures macros for the File Chooser. Table D-2 lists the attributes and macros the File Chooser. This section also lists the attributes for the `HISTORY`, `FILE_LIST` and `PATH` packages, as well as the XView Version 3.2 and newer `PANEL_LIST` and `MENU` accelerator additions. This information is described fully in the *XView Reference Manual*.

Table D-1. File Chooser Procedures and Macros

Procedures and Macros	
fchsr_case_ascend_compare()	FILE_CHOOSER_OPEN_DIALOG
fchsr_case_descend_compare()	FILE_CHOOSER_SAVE_DIALOG
fchsr_no_case_ascend_compare()	FILE_CHOOSER_SAVEAS_DIALOG
fchsr_no_case_descend_compare()	FILE_CHOOSER_NULL_COMPARE
	FILE_CHOOSER_DEFAULT_COMPARE

Table D-2. File Chooser Attributes

FILE_CHOOSER_ABBREV_VIEW	FILE_CHOOSER_FILTER_MASK
FILE_CHOOSER_APP_DIR	FILE_CHOOSER_FILTER_STRING
FILE_CHOOSER_AUTO_UPDATE	FILE_CHOOSER_HISTORY_LIST
FILE_CHOOSER_CD_FUNC	FILE_CHOOSER_MATCH_GLYPH
FILE_CHOOSER_CHILD	FILE_CHOOSER_MATCH_GLYPH_MASK
FILE_CHOOSER_COMPARE_FUNC	FILE_CHOOSER_NOTIFY_FUNC
FILE_CHOOSER_CUSTOMIZE_OPEN	FILE_CHOOSER_NO_CONFIRM
FILE_CHOOSER_DIRECTORY	FILE_CHOOSER_SAVE_TO_DIR
FILE_CHOOSER_DOC_NAME	FILE_CHOOSER_SHOW_DOT_FILES

Table D-2. File Chooser Attributes (continued)

FILE_CHOOSER_EXTEN_FUNC	FILE_CHOOSER_TYPE
FILE_CHOOSER_EXTEN_HEIGHT	FILE_CHOOSER_UPDATE
FILE_CHOOSER_FILTER_FUNC	

Table D-3. History and History Menu Attributes

HISTORY_ADD_FIXED_ENTRY	HISTORY_MENU_HISTORY_LIST
HISTORY_ADD_ROLLING_ENTRY	HISTORY_MENU_OBJECT
HISTORY_DUPLICATE_LABELS	HISTORY_NOTIFY_PROC
HISTORY_DUPLICATE_VALUES	HISTORY_ROLLING_COUNT
HISTORY_FIXED_COUNT	HISTORY_ROLLING_MAXIMUM
HISTORY_INACTIVE	HISTORY_VALUE
HISTORY_LABEL	

Table D-4. File List Attributes

FILE_LIST_ABBREV_VIEW	FILE_LIST_MATCH_GLYPH
FILE_LIST_AUTO_UPDATE	FILE_LIST_MATCH_GLYPH_MASK
FILE_LIST_CHANGE_DIR_FUNC	FILE_LIST_ROW_TYPE
FILE_LIST_COMPARE_FUNC	FILE_LIST_SHOW_DIR
FILE_LIST_DIRECTORY	FILE_LIST_SHOW_DOT_FILES
FILE_LIST_DOTDOT_STRING	FILE_LIST_UPDATE
FILE_LIST_FILTER_FUNC	FILE_LIST_USE_FRAME
FILE_LIST_FILTER_MASK	
FILE_LIST_FILTER_STRING	

Table D-5. Path Attributes

PATH_IS_DIRECTORY
PATH_USE_FRAME
PATH_RELATIVE_TO
PATH_LAST_VALIDATED

Table D-6. Version 3.2 Panel List Attributes

PANEL_LIST_INACTIVE	PANEL_LIST_MASK_GLYPHS
PANEL_LIST_DELETE_INACTIVE_ROWS	PANEL_LIST_ROW_VALUES
PANEL_LIST_DO_DBL_CLICK	PANEL_LIST_EXTENSION_DATA
PANEL_LIST_MASK_GLYPH	PANEL_LIST_EXTENSION_DATAS

Table D-7. Version 3.2 Menu Accelerator Attributes

FRAME_MENUS	MENU_ACCELERATOR
FRAME_MENU_COUNT	MENU_ACTION_ACCELERATOR
FRAME_MENU_ADD	MENU_STRINGS_AND_ACCELERATORS
FRAME_MENU_DELETE	