Integrating Applications into SCO® Open Desktop® Release 3.0

Date: 6 January 1993
Document version: 3.0.0A

APIs, standards, libraries, compatibility

This paper explains how to use the icon, display resolution, color, and session manager APIs (application programming interfaces) provided with SCO® Open Desktop®, Release 3.0.

These APIs help you integrate your application into the Open Desktop Graphical Environment. With them, you can ensure the best look for your application by defining different graphical resources for different display types. Using the Open Desktop APIs will also help make your application's appearance and behavior consistent with that of other software running on Open Desktop. Users expect and depend on such consistency, because it enables them to learn and use new applications more easily.

You may use any, all, or none of the four APIs described in this paper. We recommend you use all four, because:

- If you use the **icon object API** to put your icon on the Desktop, users can start (launch) your application by double-clicking on your icon. If you do not, users must start your application from the command line (or by finding your executable in its directory window and double-clicking on that file's icon).

- If you use the **display resources API** to adapt to display resolution at run time, Open Desktop will load those resources that give your application the best appearance on the current display. If you do not, your application may look better on some displays than on others.

- If you use the **color API** to support the Desktop color selector, users can customize your application's color scheme with the Open Desktop graphical **Color** control. If you do not, your application's colors will not change when users globally modify those of the Desktop, Desktop clients, and other applications.
- If you use the **session manager API** to communicate with the session manager, users can terminate an Open Desktop session while your application is running, then resume the session later with your application restored to the same state it was in at shutdown. If you do not, your application will still be restarted when the session is resumed, but its state at shutdown will not be restored and unsaved data may be lost.

NOTE Of the four listed above, only the icon object API is applicable to character-based applications.

What you should know

This paper assumes you are familiar with:

- Open Desktop Release 3.0
- X Window System Toolkit, Version 11, Release 5
- OSF/Motif® Toolkit, Version 1.2
- *OSF/Motif Style Guide*, Release 1.1
- ICCCM, Version 1.0 (X interclient communications protocols)
- C programming language (ANSI standard)

For more information about these systems and technologies, see the Open Desktop Development System documentation, the *Open Desktop User's Guide* and online *Desktop Help*, and the *Open Desktop Graphical Environment Administrator's Guide*. The latter also contains detailed information about Open Desktop rule and icon object script syntax.

NOTE The SCO Open Desktop Development System Release 3.0 includes X Window System® Version 11, Release 5 and the OSF/Motif Toolkit, Version 1.2. Refer to the *Release Notes* for important information about the standards and features in X11R5 and the new Motif Toolkit. Refer to the **Intro**(Xm) manual page for a description of new functionality and enhancements that affect the Motif Toolkit as a whole. See "Backward compatibility" (page 4) for information out running Open Desktop Release 3.0 binaries on Open Desktop Release 2.0.

Graphical interface standards and libraries

If your UNIX® application has a graphical interface, it must be able to run in an X Window System environment. The interface should be Motif-compliant, as defined by the *OSF/Motif Style Guide* (Revision 1.1).

Open Desktop Development System Release 3.0 is built upon Version 1.2 of the OSF/Motif Toolkit. It is possible to substantially reduce memory usage by using the X shared libraries. To take advantage of these performance improvements, link your application to the libraries listed in Table 1. Refer to the *Release Notes* for further information about X11R5 and Motif 1.2 performance improvements. Refer to the next section "Backward compatibility" (page 4) for information about running Open Desktop 3.0 binaries on Open Desktop 2.0.

Table 1 Libraries to link with

Library	Contains:
XtXm_s	Xt intrinsics and Motif routines
X11_s	X Window (Version 11) routines
Xmu	X miscellaneous utilities
c_s	shared version of a portion of libc
malloc	faster version of memory allocation routines
PW	Programmer's Workbench library needed to resolve the **alloca** symbol for the X libraries; this should be listed on the link line after XtXm_s and X11_s
socket	socket interface routines

Refer to the Development System *Encyclopedia* for more information about standard COFF libraries, also known as archive libraries, and static COFF shared libraries. For detailed information about shared libraries, see the *Programming Tools Guide*. Refer to the *Release Notes* for specific information about linking these libraries.

Backward compatibility

Applications that run on Open Desktop Release 2.0 will run on Open Desktop Release 3.0; however, the reverse may not be true. If you are building applications on Open Desktop 3.0 to run on Open Desktop 2.0, note the following.

- Use non-shared libraries to ensure portability. Note, however, that if applications are built with non-shared libraries, they will have a large binary size.
- Use shared libraries to ensure a smaller binary size. Note, however, that if applications are built with the X Window shared libraries (*libX11_s.a* and *libXtXm_s.a*), they must be run on X Window runtime shared libraries (specifically */shlib/libX11R5_s* and */shlib/libXtXm1.2_s*). X11R5 runtime shared libraries are available from the SCO X11R5 Runtime System EFS.

Applications that run on Open Desktop Release 1.1 will run on Open Desktop Release 2.0 and 3.0; however, the reverse may not be true because Release 2.0 and 3.0 add features and technology not available in Release 1.1. Specifically, the APIs discussed in this paper are new, and so not supported in releases prior to 2.0.

Release 1.1 defined icons through rules. These rules were located with other rules in rule files. When an application was installed, rule files had to be edited so the new rules could be added. The rule files had to be edited again when an application was removed or updated.

Release 2.0 and 3.0 uses a more modular approach: discrete icon objects are installed with a utility, eliminating the need to edit rule files. While icon objects simplify installation of applications onto this and future Open Desktop releases, icons defined this way will not display on Release 1.1 Desktops.

For more about installing Release 1.1-compatible applications on a Release 2.0 or 3.0 Desktop, see the "Updating Open Desktop" chapter of the *Open Desktop Installation and Update Guide*. The *Open Desktop Graphical Environment Administrator's Guide* provides administrators with instructions for transferring currently installed and configured applications from Release 1.1 to Release 2.0 or 3.0 Desktops.

Refer to the *Release Notes* for information about Motif 1.2 and Motif 1.1 binary, visual, and behavioral backward compatibility.

Notational conventions

SCO documents use font changes and other typographic conventions to distinguish text elements. The following table shows these conventions:

Font conventions

Example	Entity
cc or **cc**(CP)	command. The "CP" indicates the manual page section in which the command is documented.
passwd or *passwd*(F)	filename. The "F" indicates the manual page section in which the file is documented.
action	placeholder; replace with an appropriate value
⟨Esc⟩	keyname
$HOME	environment or shell variable
SIGHUP	named constant
"clobber"	jargon
date	user input
`Thu Feb 13 1992`	non-highlighted system output such as the output from a command
uunet	machine, network, or user name
employees	database name
Name	field name in database or named field in screen forms

Putting your icon on the Desktop

Open Desktop users expect to be able to start your application by double-clicking on its icon. If your application works with text, graphic, or other data files, users also assume that dropping a file icon on your icon will start your application working on that file.

Open Desktop uses an object-oriented approach to defining application icons. A Desktop icon is an object defined by one or more pixmap files (the icon picture), one or more object scripts (that specify what happens when users manipulate the icon), and an object directory (to hold the pixmaps and object scripts). The icon label is taken from the name of the object directory. At the beginning of a session, Open Desktop looks in designated directories for icon objects, then displays the corresponding icons.

To put your icon on the Desktop:

1. Define the icon graphic in a pixmap file.
2. Define the actions triggered by user manipulation of the icon in object script files.
3. Install icon files with the **deskconfobj** installation tool.

Detailed instructions for each step follow.

NOTE This release provides a new client, **Object Builder**, that allows you to graphically create objects. You can use **Object Builder** to perform step 2 interactively. Refer to the online *Desktop Help* and the **objbld**(X) manual page for information on using this tool.

You may also:

- Animate icons (page 18).
- Attach pop-up menus to icons (page 14).
- Provide translations of the icon label for different languages (page 21).
- Create special icons for files created by your application (page 22).

1. Create an icon pixmap

The easiest way to prepare an icon pixmap is to start with an existing icon:

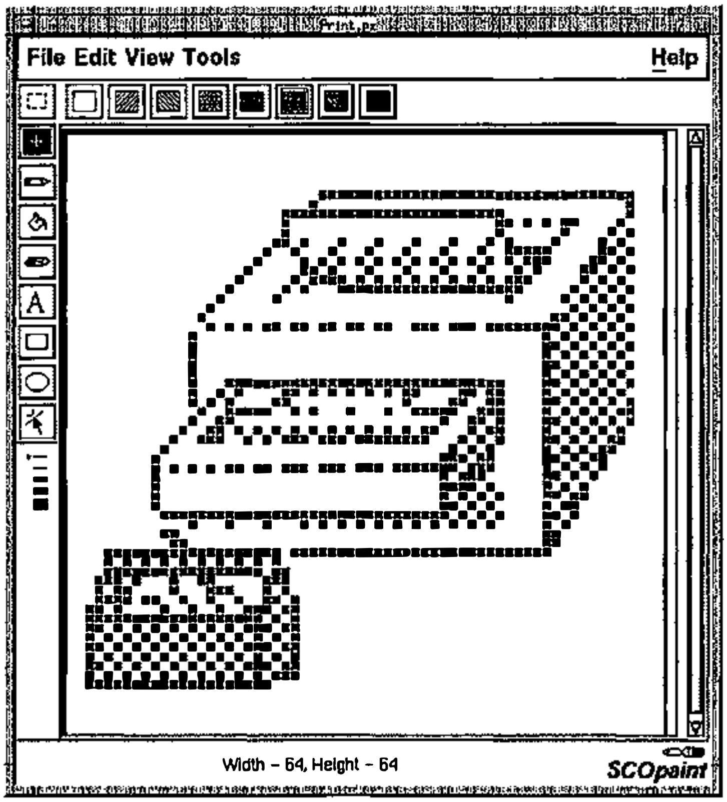
1. **Select a Desktop icon to use as a base template.**

   Choose the Desktop accessory or control icon that most resembles your icon design. Accessory icons are located in the Accessories window; control icons are in the Controls window.

2. **Drop that icon on the Paint icon.**

   This will open the **Paint** accessory with the icon you selected displayed, as shown below. (If the message `. . . does not seem to be a pixmap file. Do you want to edit the icon picture?` is displayed, click on **Yes**.)

   NOTE To edit color icons with the **Paint** accessory, you must work on a color monitor with a server that supports up to 256 colors.



We recommend you use the **Paint** accessory (located in the Desktop Accessories window) to create your icon. Although you may use any icon editor or graphics program that can save to a file in XPM2 (Version 2.8) pixmap format, only pixmaps created with the **Paint** accessory can use the transparent "color" needed to create borderless icons. Instructions for using the **Paint** accessory are available online through **Help**.

3. **Edit the image into the picture you want.**

   This technique ensures that your pixmap will be based on a 64-by-64-pixel grid, and that it uses appropriate colors.

   Open Desktop can be run on servers that support as few as 16 colors. Colors are identified by their hexadecimal values at the beginning of a pixmap file (and can be viewed with a text editor). To ensure that your icon will look good even when displayed by 16-color servers, use the same colors as other Desktop icons do:

	R G B	
icon color 1:	# FFFF F3F3 A6A6	(yellowish)
icon color 2:	# FFFF 8A8A 8686	(reddish)
icon color 3:	# 1818 C7C7 E7E7	(blueish)
black		
white		
transparent		

   When you use one of the Desktop icons as a starting point, these colors are already selected for you.

   Blend additional colors by stippling these together in various combinations. If you want to adapt an icon you have created elsewhere, use the **Paint** accessory's **Grab Color** feature to capture colors from a Desktop icon and transfer them to your icon.

   The transparent "color" lets the background color of the Desktop (or directory window) show through, so you can create borderless icons. Generally, an icon can be more readily picked out of a group of distinctly shaped icons than it can from a group of icons with identical rectangular borders.

   Do not include the icon label in the pixmap. The label is defined by the name of the icon object directory.

4. **Save the edited image.**

   Save the edited image in a pixmap file named *picture.px*.

   Note that if you have an existing icon that you had previously created in black-and-white bitmap format, you may use it, as long as you rename the bitmap file *picture.px*.

5. **Test your icon's appearance on various servers and displays.**

   Your icon will be seen on a variety of displays. View your icon on different displays to verify that it is legible in various resolutions and on grayscale displays. To get the best look on monochrome displays, use the **Paint** accessory to explicitly map each color you use to black or white. Refer to the **scopaint**(X) manual page for more information.

   To check that your icon can be recognized when it is stippled, drag it onto the Desktop from its directory window. A stippled copy of the icon will remain behind:

2. *Define icon triggers*

The action of pressing a mouse button while pointing to an icon is called an icon trigger, because it triggers an action associated with the icon. Table 2 shows the default Open Desktop icon triggers.

Table 2 Icon triggers

User action	Mouse button	Trigger type	Object script file name	Used for
double-click	1	static	s1.objscr	start
double-click	2	static	s2.objscr	alternate start
double-click	3	static	s3.objscr	start with prompt
hold down	3	hold	h3.objscr	pop-up menu
drag and drop	1	drop	d1.objscr	start on file
drag and drop	2	drop	d1.objscr	start on file
drag and drop	⟨Ctrl⟩ 1	drop	d2.objscr	alternate start on file
drag and drop	⟨Ctrl⟩ 2	drop	d2.objscr	alternate start on file
drag and drop	3	drop	d3.objscr	start on file with prompt

On a three-button mouse, the buttons are referred to as mouse buttons 1, 2 and 3. Mouse buttons 1 (the left button by default) and 3 (the right button by default) can be swapped to accommodate left-handed users. On a two-button mouse, users can simulate button 2 by pressing both buttons simultaneously. For more about how a mouse is used, see the *Open Desktop User's Guide* or *Desktop Help*.

To define an icon trigger:

1. **Write a Deskshell object script for each trigger you want to define.**

   You should only define actions for the triggers listed in Table 2. Other user manipulations of your icon (single-clicking on it or dragging it to a new location, for example) are handled automatically by Open Desktop. Sample object scripts for s1, d1, and h3 triggers are presented on the following pages.

   The s1 trigger must be used to start your application. By default, double-clicking with mouse button 3 (the s3 trigger) brings up a a dialog box that prompts the user for command line options to the startup command in your s1 script. Similarly, a file icon dragged with mouse button 3 and dropped on your icon (the d3 trigger) brings up a prompt for startup options, then executes your script for the d1 trigger. The s3 and d3 triggers prompt for options to the s1 and d1 scripts by default; you do not need to write scripts for these triggers unless you want to override their default behavior.

   Use the Deskshell command language to write object scripts. It provides the same basic functions as the Bourne shell, along with additional commands for defining special Desktop actions. Instructions for using Deskshell are in the chapter "Writing scripts in Deskshell" in the *Open Desktop Graphical Environment Administrator's Guide*.

   Only write object scripts for those triggers that are relevant to your application. For example, an application that simply displays a clock does not need to define what happens when a text file is dropped on it. If a trigger is not defined with an object script, a message will notify the user that the action (double-click, hold, or drop) is not defined for that button.

2. **Name each object script file as specified in Table 2.**

   Note that dragging and dropping with either mouse button 1 or 2 triggers the d1 object script. To trigger the d2 script, users must hold down the ⟨Ctrl⟩ key while dragging with mouse button 1 or 2.

When assigning functions to mouse buttons, follow the guidelines defined in the *OSF/Motif Style Guide* regarding which button does what. Your application will be easier to learn if it uses the mouse buttons in the same ways other applications do.

Starting your application with the s1 trigger

Your s1 object script should start your application when a user double-clicks on your icon with mouse button 1. This script must be named *s1.objscr* ("s" for "static trigger," "1" for "mouse button 1," "objscr" for "object script").

For example, an *s1.objscr* script to start a graphical clock could be as simple as:

```
/usr/bin/X11/xclock $*
```

By default, double-clicking on your icon with mouse button 3 will display a dialog box that prompts the user for command line options. Open Desktop then executes the *s1.objscr* script with the options the user enters. These options could be standard Xt intrinsics options or options specific to your application. In the above example, **$*** includes any options entered in the dialog box.

Be careful about using relative pathnames in your object scripts. Your scripts will be installed with the **deskconfobj** tool, and you cannot assume you know in which directory they will be placed.

To start a character-based application with its icon, the s1 script must open a DOS or UNIX (**scoterm**) window in which to run the application. The following Deskshell object script starts a UNIX character-based program named **calendar** in a **scoterm** window.

```
cd $HOME
shell -n 'Calendar' calendar $*
```

Because this application will be creating files, the first line changes the current directory to the user's home directory. Otherwise, the user would be unpleasantly surprised to find that the files were created in the **calendar** icon's object directory. **shell** is a Desktop resource that defines the terminal emulation; it is set to **scoterm** by default. The **-n** option specifies a title for the **scoterm** window.

For more about the UNIX terminal window, see the **scoterm**(X) manual page. For more about the DOS window, see the "Administering DOS Services" section of the *Open Desktop Graphical Environment Administrator's Guide*.

Starting your application with the d1 trigger

If your application uses text, graphic, or other data files, users expect that dropping a file icon on your icon will start your application working on that file. For example, dropping a text file on an editor's icon should start the editor with the text file open and ready to edit.

The following one-line d1 object script starts a text editor and passes to the editor the name of the file dropped on it. Open Desktop sets the variable **$dynamic_args** to the filename(s) represented by the dropped icon(s). **$*** includes any options entered in the dialog box displayed by the d3 trigger. (For more about Desktop variables, see the *Open Desktop Graphical Environment Administrator's Guide*.)

```
editor $* $dynamic_args
```

Users can drop a group of icons on your application's icon. Consider how your application will handle this. For example, should the text editor open an editor window for each icon dropped? Or should it open one file and place the others in a queue? If more than one approach could be useful, let users choose the default behavior.

Popping up a menu with the h3 trigger

Only button 3 can be used as a hold trigger. If defined, the h3 trigger must display a pop-up menu.

For example, this one-line *h3.objscr* script pops up the menu **calendar_menu** (which is used to start **calendar** with different startup options):

```
calendar_menu
```

The menu itself is defined by a rule, such as the following, which would be included in your application rules file and installed with the **deskconfobj** tool (page 16). For more about Desktop rules, see the *Open Desktop Graphical Environment Administrator's Guide*.

```
 1    menu: calendar_menu
 2    {
 3      menu_item: Daily
 4      {
 5        calendar -daily
 6      }
 7      menu_item: Monthly
 8      {
 9        calendar -monthly
10      }
11      menu_item
12      {
13        title=Master;
14        enable_if
15        {
16          [ == $USER root ]
17        }
18        select_action
19        {
20          calendar -master
21        }
22      }
23    }
```

lines 3-10	Define the **Daily** and **Monthly** menu items. Each item causes the **calendar** executable to be run with the appropriate option.
lines 11-23	Define the **Master** menu item. Because the master calendar is only available to superusers, these lines test whether the user is logged in as *root*. If not, the menu item is dimmed (stippled) to indicate it is not available to the current user.

3. Install icon files

The approach to icon definition depends on correct installation of your pixmap, object scripts, and rule files. Use the **deskconfobj** object installation tool in your installation script to ensure that your icon files are installed properly. Your installation script tells **deskconfobj** what to install, and **deskconfobj** determines where.

In this release, pixmaps and object scripts are copied into a directory named *label.obj*, where ***label*** is the label that appears beneath your icon. This directory is created in the */usr/lib/X11/XDesktop3/applications* directory. Application rules are currently installed in the */usr/lib/X11/XDesktop3/C.xdt/apprules* directory or in the */usr/lib/X11/XDesktop3/**lang**.xdt/apprules* directory, depending on whether you install localized versions of your rules. These directories may change in future releases, but if you use **deskconfobj** to install your icon object, your installation script will still work.

Because ordinary users usually do not have permission to copy files into the required directories, **deskconfobj** can only be used if the installing administrator is logged in as *root*. If you want your application to be installable by users in their directories, see "Making applications user-installable" (page 18).

NOTE If you will be installing your application with the SCO **custom**(ADM) installation utility, put the following commands in your **init** scripts.

In your installation script:

1. **Reset the file-creation mode mask with:**

   ```
   umask 002
   ```

2. **Temporarily add** */usr/lib/X11/XDesktop3* **to the search path.**

   Alternatively, you can prepend this path to each **deskconfobj** command.

3. **Install icons and object scripts with this syntax:**

   ```
   deskconfobj -aobj -name label -src source_directory
   ```

   Use the -aobj (add object) option to install a new object on the Desktop. For ***label***, substitute the text that will be displayed beneath your icon. You may use the name of your application for the label, or any other string that is allowed in a UNIX directory name. For ***source_directory***, substitute the path to the directory from which icon pixmaps and object scripts are to be copied. All files in the source directory will be copied into the */usr/lib/X11/XDesktop3/applications/**label**.obj* directory.

   (If an error message indicating that **deskconfobj** cannot copy filesappears, you may safely ignore it. This is a known bug.)

4. **Install application rules (if any) with this syntax:**

   ```
   deskconfobj -arule -name rule_name -src file [-lang language]
   ```

   Use the -arule (add rule) option to add a rules file associated with your object. For ***rule_name,*** substitute a name that is unlikely to be duplicated by other programs (typically your application's class name). For ***file,*** substitute the pathname of the file that contains the rules to be copied. Most applications should put all of their application rules in a single source file (or in a series of localized source files with the same name).

   The environment variable **$LANG** can be set to indicate a user's language preference. To install localized application rules, use the **-lang** option. For ***language,*** substitute the appropriate value of **$LANG**. Repeat the command for each language you will support.

   Your rule file will be copied into the */usr/lib/X11/XDesktop3/C.xdt/apprules* directory if you do not specify a language, and into the */usr/lib/X11/XDesktop3/lang.xdt/apprules* directory if you do. Even if you only support an English version of your rules, you should also install it with the **-lang english** option. Many systems are configured with English as the default language. For more about **$LANG** and localization, see the Open Desktop Development System documentation.

   NOTE Newly installed rules will not be recognized until Open Desktop is restarted.

5. **Remove** */usr/lib/X11/XDesktop3* **from the search path.**

   Skip this step if you prepended this path to the commands instead of adding it to the search path in step 1.

Removing pixmaps, object files, and application rules

You may also use **deskconfobj** in your "de-installation" script to cleanly remove files it has installed and icon object directories it has created.

To remove an icon object, use this syntax:

```
deskconfobj -dobj -name label
```

This will remove the entire icon object, including the *label.obj* directory and all the pixmaps and object scripts in it.

To remove an application rule file, use this syntax:

```
deskconfobj -drule -name rule_name [ -lang language ]
```

Use the **-lang** option to remove a localized application rule file.

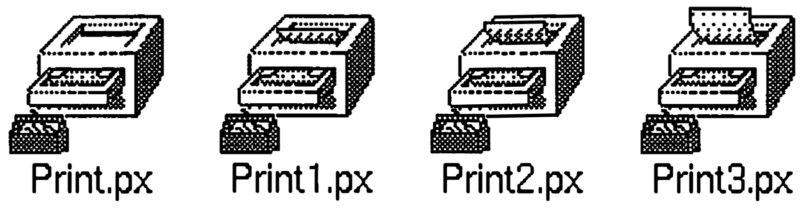
Making applications user-installable

The **deskconfobj** tool cannot be used to install an application into a user's directory. You can install an icon object in a user's directory tree by creating an *label.obj* directory and copying your pixmap(s) and object script(s) into it. Your icon will appear in the *label.obj* directory's parent directory's window.

In Open Desktop, rules specific to a directory are installed in a directory rules file named *.xdtdirinfo*. Merge your application rule file into the *.xdtdirinfo* file in the parent directory of the *label.obj* directory. (Create the *.xdtdirinfo* file if it does not already exist.) To avoid conflicts with rules already in the *.xdtdirinfo* file, it is best to let the user to modify the file by hand. For more about *.xdtdirinfo* files, see the *Open Desktop Graphical Environment Administrator's Guide.*

Animating icons

You may animate your icon by presenting a series of icon pictures in sequence. The **Print** icon on the Desktop is an animated icon; when a file icon is dropped on it, paper slides out of the printer. The **Print** icon is animated by stepping through this series of pictures:



You can define an animated icon just like any other icon, except you must supply more than one pixmap, along with directions for when to display each pixmap.

To create an animated icon:

1. **Create the pixmaps.**

   For more about creating icon pixmaps, see "Create a icon pixmap" (page 8).

   The example used to illustrate these instructions defines an appointment book icon that opens when a **calendar** program is started. This icon picture is made of four pixmaps: closed book (*picture.px*), book one-third open (*Calendar1.px*), book two-thirds open (*Calendar2.px*), and open book (*Calendar3.px*).

2. **Write a rule that assigns each file to a variation class.**

   Variation classes identify alternate definitions of your icon object.

   For example:

```
 1    icon_rules
 2    {
 3      Calendar.obj      /DRXØ
 4      {
 5        picture=%PØ/picture.px
 6      }
 7      Calendar.obj      /DRX1
 8      {
 9        picture=%PØ/Calendar1.px
10      }
11      Calendar.obj      /DRX2
12      {
13        picture=%PØ/Calendar2.px
14      }
15      Calendar.obj      /DRX3
16      {
17        picture=%PØ/Calendar3.px
18      }
19    }
```

lines 3-6	Specify that for variation 1 of *Calendar.obj*, the picture (pixmap) is *Calendar1.px*. **/DRX** specifies that the named Directory is Readable and eXecutable, and the number following it identifies the variation class. **%PØ/** instructs Open Desktop to substitute the absolute path of the icon object directory.
lines 7-17	Specify pixmaps for variation classes 2 and 3.

Put the rule in the same file with your other Desktop rules (if any). For more about Desktop rules, see the *Open Desktop Graphical Environment Administrator's Guide.*

3. **Add variation class declarations to the *s1.objscr* script.**

   For example:

```
1      vclass 1 $static_arg
2      check $static_arg                  # display Calendar1.px
3      vclass 2 $static_arg
4      check $static_arg                  # display Calendar2.px
5      vclass 3 $static_arg
6      check $static_arg                  # display Calendar3.px

7      /usr/bin/X11/calendar -monthly $*

8      vclass 0 $static_arg
9      check $static_arg
```

lines 1-6	Step through the variation classes, displaying the pixmap for each in sequence. **vclass** sets the variation class of the **$static_arg** (the icon object). **check** causes Open Desktop to update the pixmap display. Because the variation class 0 pixmap (by default, *picture.px*) is already displayed when this script is triggered, the sequence begins with it.
line 7	Starts the application. **$*** includes any options the user enters in the dialog box displayed by the s3 trigger (page 12).
lines 8-9	Resets the variation class to 0 and redisplays the *picture.px* pixmap.

   For more about object scripts, see "Define icon triggers" (page 11).

   Animation can also be used to convey information about the state of your application. For example, an icon for a mail program could change its appearance to indicate that new mail has arrived. If you want to indicate the state change when an icon trigger is activated, test for the state and declare the variation class in the appropriate object script. If you want to indicate the state change without waiting for a trigger, your application must test for the state change and send instructions directly to the Desktop (see the **tellxdt3**(X) manual page and the *Open Desktop Graphical Environment Administrator's Guide*).

4. **Install the pixmaps, rules, and object scripts with the deskconfobj tool.**

   The pixmaps and scripts will be installed in the icon object directory. By default, your rule file will be copied into the */usr/lib/X11/XDesktop3/C.xdt/apprules* directory. If you install localized versions of your rules, your rule files will be installed in the */usr/lib/X11/XDesktop3/lang.xdt/apprules* directory. For more about the **deskconfobj** tool, see "Install icon files" (page 16).

Localizing icon labels

Your icon's label is taken from the name of your icon's object directory. If the directory is named *Calendar.obj*, by default the icon will be labeled "Calendar". The following explains how to override the default label.

The environment variable **$LANG** can be set to indicate a user's language preference. You can localize your icon label by writing rules that define a different label for each setting of **$LANG**. In the following example, the **calendar** icon's label reads "Calendario" when **$LANG** is set to **spanish**. For more about **$LANG** and localization, see the Open Desktop Development System documentation.

To localize your icon's label:

1. **Write a rule that defines the label for each language you support.**

   For example, this rule defines a Spanish label for the **calendar** icon:

```
1       icon_rules
2       {
3          Calendar.obj       /DX
4          {
5           title=Calendario;
6          }
7       }
```

line 3	**/DX** specifies that the named Directory is eXecutable.
line 5	Explicitly defines the icon label. Without such a definition, the label is taken from the name of the icon object directory. The label can be any string that is allowed in a UNIX directory name.

   For more about Desktop rules, see the *Open Desktop Graphical Environment Administrator's Guide*.

2. **Use the deskconfobj tool's -lang option to install your rule in a language-specific directory.**

   For example, the following syntax will install the sample rule above in the */usr/lib/X11/XDesktop3/spanish.xdt/apprules* directory:

```
deskconfobj -arule -name Calendar -src ./CalRules -lang spanish
```

   For more about using **deskconfobj**, see "Install icon files" (page 16).

Creating icons for user data files

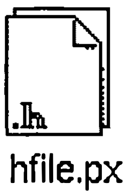
You can use application rules to define special icons for certain types of files, and to start your application when a user double-click on one of those icons. For example, specially formatted calendar files created and displayed by a **calendar** program could be represented by a calendar file icons.

To define special icons for data files:

1. **Create a pixmap for the file icon.**

   The name of the pixmap file must end in the *.px* extension. For more about creating icon pixmaps, see "Create a icon pixmap" (page 8).

   To help users identify it as a *file* icon, you should base the design of your data file icon on the standard Open Desktop file icon. For example, a C header file is represented by a file icon labeled with an ".h":



2. **Write a rule that assigns the icon to a type of file.**

   For example, the following rule specifies that files with the *.cal* suffix will be represented by the icon pixmap file *Cal_file.px*. The rule also specifies that double-clicking on one of these file icons will have the same effect as dropping the file icon on the application's icon: it will start the **calendar** program with the calendar file displayed. For more about Desktop rules, see the *Open Desktop Graphical Environment Administrator's Guide*.

```
 1 icon_rules
 2 {
 3   *.cal /F
 4   {
 5     picture=Cal_file.px;

 6     trigger_action: s1
 7     {
 8       filetype /usr/bin/calendar
 9       if [ == $_type(1) F ] && [ == $_type(2) X ]
10       then
11         /usr/bin/calendar $* $static_arg
12       else
13         fyi -t 'Calendar' 'Calendar is not installed on your system'
14       fi
15     }
16   }
17 }
```

line 1	Identifies the start of the **icon_rule** section of your rule file.
lines 3-5	Specify that files with names ending in *.cal* will display the *Cal_file.px* pixmap. Open Desktop searches certain directories below */usr/include/X11/bitmaps* for the file.
lines 6-15	Define what happens when the user double-clicks (the s1 or d1 trigger) on the file icon.
lines 8-9	Check the file type of */usr/bin/calendar*. If it is an executable, do line 11. If not, do line 13.
line 11	Executes the **calendar** program. **$*** includes any arguments the user enters in the d1 trigger dialog box. The filename(s) represented by the dropped icon(s) are substituted for **$static_arg**.
line 13	Prints an error message if **/usr/bin/calendar** is not executable. The message assumes that not having execute permission for the file is, for the Desktop user, functionally equivalent to its not being installed.

3. **Use the deskconfobj tool to install your pixmap and rule.**

   Use the following syntax to install your pixmap:

   ```
   deskconfobj -alrg -name pixmap_name -src file
   ```

   Use the -alrg (add large pixmap) option to add a pixmap. For ***pixmap_name,*** substitute the name of the pixmap file (*Cal_file.px* for the above example). For ***file,*** substitute the pathname of the pixmap file to be copied.

   Use the following syntax to install your rule:

   ```
   deskconfobj -arule -name rule_name -src file
   ```

   For ***rule_name,*** substitute a name that is unlikely to be duplicated by other programs (typically your application's class name). For ***file,*** substitute the pathname of the file that contains the rules to be copied.

   NOTE Newly installed rules will not be recognized until Open Desktop is restarted.

   The data file's name will be used as the icon label. For more about using **deskconfobj**, see "Install icon files" (page 16).

Adapting to display resolution at run time

Open Desktop is designed to run on a wide variety of hardware platforms, with resolutions ranging from 640-by-480 pixels to beyond 1280-by-1024 pixels. Visual resources defined for a specific display resolution can produce inappropriately sized, unpleasant, or even illegible results when your application is displayed at a different resolution.

You may specify a different set of visual resources for each display resolution or type with an Open Desktop ODR (Open Desktop display resources) file. In an ODR file, standard **#if, #elif,** and **#else** preprocessor directives are used to test the values of certain display attributes, such as WIDTH. For example, the following tests the WIDTH of the display (in pixels) and specifies font sizes accordingly.

```
#if WIDTH <= 800
  TypicalApp*create_button.FontList      -*-helvetica-bold-r-*--10-*-p-*
  TypicalApp*delete_button.FontList      -*-helvetica-bold-r-*--10-*-p-*
#elif WIDTH <= 1024
  TypicalApp*create_button.FontList      -*-helvetica-bold-r-*--12-*-p-*
  TypicalApp*delete_button.FontList      -*-helvetica-bold-r-*--12-*-p-*
#elif WIDTH <= 1280
  TypicalApp*create_button.FontList      -*-helvetica-bold-r-*--14-*-p-*
  TypicalApp*delete_button.FontList      -*-helvetica-bold-r-*--14-*-p-*
#else
  TypicalApp*create_button.FontList      -*-helvetica-bold-r-*--18-*-p-*
  TypicalApp*delete_button.FontList      -*-helvetica-bold-r-*--18-*-p-*
#endif
```

See the Development System *Encyclopedia* for more information about preprocessors.

At the beginning of an Open Desktop session, the session manager uses **xrdb**, the X resource database utility, to check display attributes and resolve the conditional statements in your ODR file. In this way, visual resources appropriate to each display are loaded into the RESOURCE_MANAGER property of the root window.

Although you may define resources for as many display resolutions as you like, we recommend that you at least support the following four resolutions (WIDTH x HEIGHT in pixels):

640 x 480	(VGA)
800 x 600	(SVGA)
1024 x 768	(SVGA)
1280 x 1024	(high-resolution)

Most applications will only need to specify resolution-dependent resources in ODR files, but you can test for any display attribute recognized by the **xrdb** utility (see Table 3). If your application does not use the Open Desktop color API (page 29), you may want to define color resources in your ODR file (page 32).

You do not need to define any of the default Open Desktop color or font resources unless you want to override them. The default font and color resources are defined in */usr/lib/X11/sco/startup/Fonts* and */usr/lib/X11/sco/startup/Colors.*

For more about graphical resources, see the *Graphical Environment Administrator's Guide*. Remember that your resource definitions only affect the content of the window in which your application is running. The window's frame and decorations are controlled by the window manager's resources.

To specify resolution-dependent resources:

1. **Separate resolution-dependent resources from other visual resources.**

   Because the ODR files for all installed applications are loaded at the beginning of a session, only define display-dependent resources in ODR files. Define other resources in an *app-defaults* file, as usual (for more about *app-defaults* files, see the *Open Desktop Graphical Environment Administrator's Guide*).

2. **Define resolution-dependent resources in an ODR file.**

   Define all of your display-dependent resources in a single ODR file. Choose a name for your ODR file that is unlikely to be duplicated by other programs (typically your application's class name). Within this file, organize resources according to the type of display with which they are to be used. Use standard **#if**, **#elif**, and **#else** preprocessor directives and **xrdb** display attribute symbols to conditionally specify resources. We use the WIDTH (screen width in pixels) symbol to define resolution-dependent resources.

Table 3 Display attribute symbols

Symbol	Definition
WIDTH	screen width in pixels
HEIGHT	screen height in pixels
X_RESOLUTION	horizontal screen resolution in pixels/meter
Y_RESOLUTION	vertical screen resolution in pixels/meter
CLASS	GrayScale, PseudoColor, TrueColor, DirectColor, StaticColor, or StaticGray
COLOR	defined only if CLASS = PseudoColor, TrueColor, DirectColor, or StaticColor
PLANES	display depth in bit planes; this value defines the number of available colors
BITS_PER_PLANE	number of significant bits in an RGB color specification; usually unrelated to PLANES

For more about display attributes symbols, see the **xrdb**(X) manual page.

To avoid conflicts with other applications' ODR resources loaded at the same time, begin the name of each resource in your ODR file with your application's class. For example, an application that displays a calendar might include the following in its ODR file:

```
#if WIDTH · 800
        Calendar.main.width:                    300
        Calendar.main.height:                   200
#elif WIDTH · 1024
        Calendar.main.width:                    400
        Calendar.main.height:                   300
#elif WIDTH · 1280
        Calendar.main.width:                    500
        Calendar.main.height:                   400
#else
        Calendar.main.width:                    700
        Calendar.main.height:                   500
#endif
```

3. **Install your ODR file in the** ***/usr/lib/X11/sco/startup*** **directory.**

   Do this in your installation script.

   At the start of an Open Desktop session, the session manager will use the **xrdb** utility to process your ODR file and load the appropriate resources.

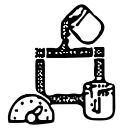
4. **Define default display resources in your** ***app-defaults*** **file.**

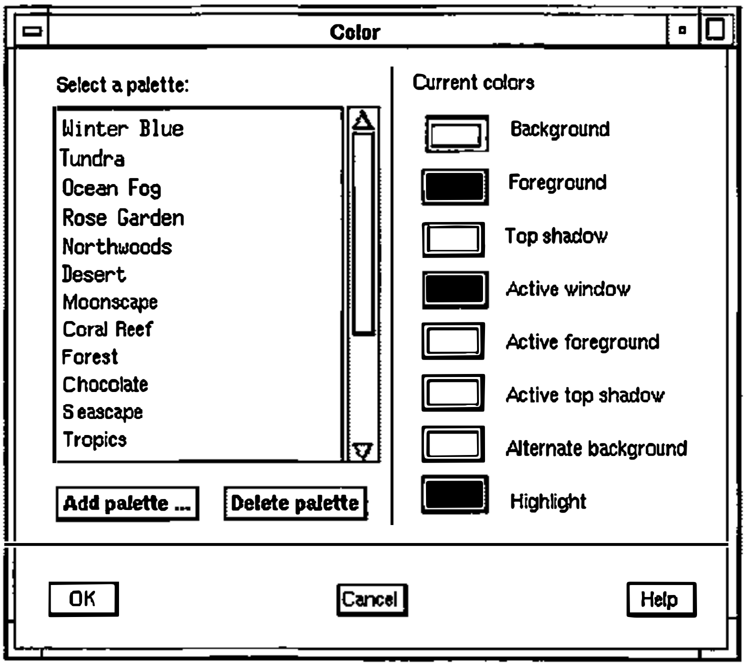
   To ensure that your application will always display, even if it is run without the session manager (and, therefore, without your ODR file loaded), define default resources in your *app-defaults* file. Include resource definitions for a 1024-by-768-pixel display (the resolution most widely used with Open Desktop) from your ODR file. Also include the default font resource definitions from */usr/lib/X11/sco/startup/Fonts* and the default color resource definitions from */usr/lib/X11/sco/startup/Colors*. Place these default definitions before any overriding definitions of your own. Name your *app-defaults* file with your application's class name, and install it in the */usr/lib/X11/app-defaults* directory.

   When they *are* loaded, your ODR file resource definitions take precedence over the defaults defined in your *app-defaults* file.

Supporting the Desktop color selector

Open Desktop users can use the Desktop **Color** control utility to simultaneously modify the coloring of all Desktop utilities, window frames, menus, and backgrounds. If you specify your color resources through the Open Desktop color API, users will be able to personalize your application's colors along with those of the rest of the Desktop.

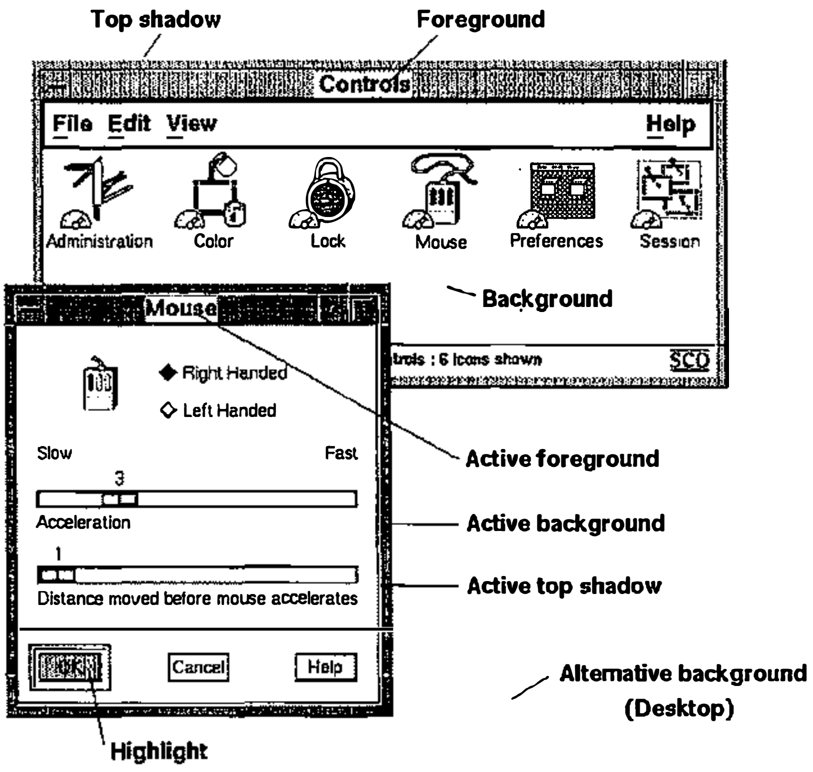

Color



The color API is based on palettes. A palette is a coordinated set of eight colors, defined by the current values of the eight Open Desktop color name variables. Each variable defines the color for several graphical objects.

Table 4 Color name variables

Variable	Typically used for
scoBackground	window frames, window backgrounds, scrollbar troughs
scoAltBackground	Desktop background, second background color
scoForeground	text
scoTopShadow	top shadows of frames, buttons and other objects
scoActiveBackground	frame of active window
scoActiveForeground	text in frame of active window
scoActiveTopShadow	top shadows in active window
scoHighlight	text or objects that need special emphasis, pressed buttons



The active window is the window with the input focus. The variables scoActiveBackground, scoActiveForeground, scoActiveTopShadow are used to distinguish the active window from other windows. In the supplied palettes, scoTopShadow is lighter than scoBackground, and scoBackground contrasts with scoForeground. Bottom shadows are always black. When using the color name variables for interface elements other than those listed, consider the relationships of the colors and how that might affect contrast or legibility.

Color palettes are automatically mapped to grayscale displays. Because this might not always yield optimal results, several grayscale palettes are provided. A default monochrome palette mapping is used on monochrome displays.

For more information about the Open Desktop **Color** control, see the **scocolor**(X) manual page and the online *Desktop Help*. For more about Open Desktop color resources, see the *Open Desktop Graphical Environment Administrator's Guide*.

To support the Desktop color selector:

1. **Use Open Desktop color name variables to define color resources.**

   In your *app-defaults* file, define color resources with the color name variables listed in Table 4. Open Desktop will set the variables to the corresponding color definitions from the currently selected palette.

   For example, an application that displays a calendar might include the following resource definitions in its *app-defaults* file:

   ```
   *datebox.back.background:          scoAltBackground
   *current.week.background:          scoActiveForeground
   *meeting.alarm.background:         scoHighlight
   ```

   Name your *app-defaults* file with your application's class name, and install it in the */usr/lib/X11/app-defaults* directory. For more about *app-defaults* files, see the *Open Desktop Graphical Environment Administrator's Guide*.

2. **Link your application with Release 3.0 of the Open Desktop Development System.**

   Color name variables used by applications that have *not* been linked with Release 3.0 will be mapped to the default palette, Winter Blue. When a user changes Desktop colors by selecting a new palette with the **Color** control, your application's colors will continue to be defined by the Winter Blue palette. For more about linking, refer to the Development System *Encyclopedia*.

3. **Include the default color resources in your *app-defaults* file.**

   To ensure that your application will always display, even if it is run without the session manager, copy into your *app-defaults* file the default color resource definitions from */usr/lib/X11/sco/startup/Colors* (listed below). Place these definitions before any overriding definitions of your own.

```
*Background:                    scoBackground
*Foreground:                    scoForeground
*topShadowColor:                scoTopShadow
*bottomShadowColor:             Black
*activeBackground:              scoActiveBackground
*activeForeground:              scoActiveForeground
*activeTopShadowColor:          scoActiveTopShadow
*activeBottomShadowColor:       Black
*troughColor:                   scoBackground
*armColor:                      scoHighlight
*highlightColor:                scoForeground
*selectColor:                   scoForeground
*borderColor:                   Black
```

Defining display-dependent colors

If your application makes extensive or special use of color, you may want to define your color resources directly rather than through the color API's color name variables.

NOTE The following applies only to those color resources that are not defined with the Open Desktop color API color name variables (page 29).

When defining color resources directly, consider the wide variety of displays on which Open Desktop is viewed. Resources defined for a color display may not work for a monochrome or grayscale display, and some servers support more colors or shades of gray than do others.

If your colors work well on all supported displays (color, monochrome, grayscale) and servers (16-color and above), define your color resources in your *app-defaults* file. If you want to use different colors for different displays or servers, conditionally define your color resources in your ODR file. For more about preparing an ODR file, see "Adapting to display resolution at run time" (page 25).

Define color resources in your ODR file *only* if your application makes special use of color that cannot be achieved with the color name variables. You may explicitly define colors for certain objects and use the color API for the rest.

This approach is useful when you want to prevent users from changing the color of an object (the red, green, and blue slider bars of an RGB color mixer, for example).

Define display-dependent resources as you would resolution-dependent ones: use standard **#if**, **#elif**, and **#else** preprocessor directives and **xrdb** display attribute symbols (page 27) to conditionally specify resources.

Use the **xrdb** COLOR symbol to differentiate color and monochrome displays.

```
#if COLOR
        . . .
        [definitions for color displays]
        . . .
#else
        . . .
        [definitions for monochrome displays]
        . . .
#endif
```

For most applications, defining separate resources for color and monochrome displays should be adequate. Check how your color resources look on a grayscale display. If the result is unacceptable, use the CLASS symbol to define grayscale resources separately.

```
#if CLASS = GrayScale
        . . .
        [definitions for grayscale displays]
        . . .
#elif COLOR
        . . .
        [definitions for color displays]
        . . .
#else
        . . .
        [definitions for monochrome displays]
        . . .
#endif
```

NOTE Open Desktop may be running on a server that only supports 16 colors (or shades of gray). On 16-color servers, only three color cells are available for definition by applications. The rest are taken by the palette manager (eight color cells), the server (two color cells, for black and white), and the Desktop icons (three color cells). You may use all 16 colors, but you may only define up to three. If you need to define more colors, you must install your own colormap using the facilities provided by the Xlib library.

Use the PLANES symbol to specify resources for 16-color (or grayscale) servers.

```
#if CLASS = GrayScale
        #if PLANES < 8
                . . .
                [definitions for 16-grayscale displays]
                . . .
        #else
                . . .
                [definitions for 256(or more)-grayscale displays]
                . . .
        #endif
#elif COLOR
        #if PLANES < 8
                . . .
                [definitions for 16-color displays]
                . . .
        #else
                . . .
                [definitions for 256(or more)-color displays]
                . . .
        #endif
#else
        . . .
        [definitions for monochrome displays]
        . . .
#endif
```

In your *app-defaults* file, define default values for resources defined in your ODR file. These default definitions will ensure that your application will always display, even if it is run without the session manager (and, therefore, without your ODR file loaded).

Communicating with the session manager

Open Desktop users expect to be able to end a session without first terminating applications. When the session is resumed, users expect to see applications exactly where they were at shutdown. The Open Desktop session manager is responsible for gracefully shutting down all applications at the end of an Open Desktop session, then restarting them when a new session begins. Your application should supply the session manager with the information needed to restart it in the state it was in when the user logged out.

Restart information is conveyed to the session manager through the **WM_COMMAND** property placed on an application's top-level window. Before ending a session, the session manager will send a **WM_SAVE_YOURSELF** message to applications. When your application receives this message, it must immediately prepare itself for shutdown by placing itself in an appropriate state, then setting the **WM_COMMAND** property to a command that will restart the application and restore it to as similar a state as possible. No interaction with the user is permitted during this process. When the next session begins, the session manager will restart your application with the specified command. It will also restore the application's geometry and window state (iconified or maximized) if the application does not.

For example, after receiving a **WM_SAVE_YOURSELF** message, a text editor might save edits made to the file *letter* since the last user-initiated save in a temporary file named *letter.tmp*. It could then use **XSetCommand**(X) to set **WM_COMMAND** to something like **editor -rt letter.tmp**. When the session is resumed, the session manager will restart the editor with this command. The editor has defined the **-rt** option to read in the indicated temporary file, and perhaps display a message that there are unsaved edits left over from the previous session. Note that the editor could not ask the user whether it should save the unsaved edits, because no interaction with the user is permitted between receiving the **WM_SAVE_YOURSELF** message and shutdown.

If your application does not communicate with the session manager, the session manager will simply disconnect your application from the server at shutdown. Unsaved data may be lost. When the session is resumed, your application will be restarted with the command with which it was last started, and with the geometry and window state (iconized or maximized) it had at shutdown. Its state at shutdown will not be restored.

The session manager will restore a DOS or UNIX terminal window with the command that opened it. If a character-based application was started inside one of these windows after the window was opened, the application will not be restored. If the application was started as an argument to the **scoterm** command, it will be restarted with that same argument. The application's state at shutdown will not be restored.

To communicate with the session manager:

1. **Use standard Xt intrinsics calls to start your application.**

   When you use standard Xt intrinsics calls to start your application, the properties **WM_COMMAND** and **WM_CLIENT_MACHINE** are placed on your top-level window, **WM_COMMAND** is set to your startup command, and **WM_CLIENT_MACHINE** is set to the name of the machine on which your application is running.

2. **Place the atom WM_SAVE_YOURSELF in the WM_PROTOCOLS property on your application's top-level window.**

   To do this, include the following in your application's code.

```
 1 #include <X11/Intrinsic.h>
 2 #include <X11/Protocols.h>
 3 #include <X11/AtomMgr.h>
 4 #include <X11/Xatom.h>

 5 void
 6 SetupSMProtocols(toplevel)
 7 Widget  toplevel;
 8 {
 9          extern  void    SaveYourselfCB( );

10          Atom    protocols[1];
11          Atom    xa_WM_SAVE_YOURSELF;

12          xa_WM_SAVE_YOURSELF = XmInternAtom(XtDisplay(toplevel),
13                                      "WM_SAVE_YOURSELF", FALSE);
14          protocols[0] = xa_WM_SAVE_YOURSELF;
15          XmAddWMProtocols(toplevel, protocols, 1);

16          XmAddWMProtocolCallback(toplevel, xa_WM_SAVE_YOURSELF,
17                                  SaveYourselfCB, NULL);
18 }
```

lines 12-15	Include the **WM_SAVE_YOURSELF** atom in the **WM_PROTOCOLS** property on your application's top level window.
lines 16-17	Set up the callback to respond to the session manager's shutdown notification.

3. **Prepare for shutdown when notified.**

   The **SetupSMProtocols(toplevel)** routine defined in the previous listing sets up the callback with which your application responds to **WM_SAVE_YOURSELF**. Include the following in your application's code to define the callback routine.

```
 1 void
 2 SaveYourselfCB(toplevel, client_data, call_data)
 3 Widget          toplevel;
 4 XtPointer       client_data;
 5 XtPointer       call_data;
 6 {
 7   char             *cmd_argv[1];

 8   cmd_argv[0] = GetBinaryFullPathName( );

 9   XSetCommand(XtDisplay(toplevel), XtWindow(toplevel), cmd_argv, 1);
10 }
```

lines 2-5	**toplevel** is your application's top level widget. **client_data** is callback client data. **call_data** is callback data. The latter two are not used in this example.
line 8	You should substitute the full pathname of your binary for **GetBinaryFullPathName()**. Include any command line options or special geometry information needed for the restart command.
line 9	Sets the **WM_COMMAND** property.

Because the session manager will be waiting for a **PropertyNotify** from the property **WM_COMMAND** as a confirmation that your application has saved its state, it is important that **WM_COMMAND** is updated even if its contents are correct. After updating **WM_COMMAND**, your application must wait to be shut down by the session manager. If your application receives a mouse or keyboard event after responding to **WM_SAVE_YOURSELF**, the shutdown has been canceled and your application may resume operation.

Index

SCO®
OPEN SYSTEMS SOFTWARE

Please help us to write computer manuals that meet your needs by completing this form. Please post the completed form to the Publications Manager nearest you: The Santa Cruz Operation, Ltd., Croxley Centre, Hatters Lane, Watford WD1 8YN, United Kingdom; The Santa Cruz Operation, Inc., 400 Encinal Street, P.O. Box 1900, Santa Cruz, California 95061, USA or SCO Canada, Inc., 130 Bloor Street West, 10th Floor, Toronto, Ontario, Canada M5S 1N5.

Volume title: ______________________________
(Copy this from the title page of the manual)

Product: ______________________________
(for example, SCO UNIX System V Release 3.2 Operating System Version 4.0)

How long have you used this product?

❑ Less than one month ❑ Less than six months ❑ Less than one year
❑ 1 to 2 years ❑ More than 2 years

How much have you read of this manual?

❑ Entire manual ❑ Specific chapters ❑ Used only for reference

	Agree				*Disagree*
The software was fully and accurately described	❑	❑	❑	❑	❑
The manual was well organized	❑	❑	❑	❑	❑
The writing was at an appropriate technical level (neither too complicated nor too simple)	❑	❑	❑	❑	❑
It was easy to find the information I was looking for	❑	❑	❑	❑	❑
Examples were clear and easy to follow	❑	❑	❑	❑	❑
Illustrations added to my understanding of the software	❑	❑	❑	❑	❑
I liked the page design of the manual	❑	❑	❑	❑	❑

If you have specific comments or if you have found specific inaccuracies, please report these on the back of this form or on a separate sheet of paper. In the case of inaccuracies, please list the relevant page number.

May we contact you further about how to improve SCO documentation? If so, please supply the following details:

Name ______________ *Position* ______________

Company ______________________________

Address ______________________________

City & Post/Zip Code ______________________________

Country ______________________________

Telephone ______________ *Facsimile* ______________