Development Environment Transition Issues

4

If you want to port an OPEN LOOK or Motif application to the Solaris CDE desktop, you need to consider the development environment transition as well as the graphical user interface (GUI) transition. The OpenWindows and Solaris CDE development environments are different in many ways but similar in others. This chapter compares and contrasts the two development environments..

Features Exclusive to CDE	*23*
OpenWindows Versus Solaris CDE Development Environments	*24*

The Solaris CDE desktop is based on the same X server as is part of the OpenWindows environment. For example, you can access Display PostScript™ (DPS), and the XIL™, and XGL™ libraries from Solaris CDE.

However, the X server is considered part of the OpenWindows development environment whereas it is *not* part of the CDE development environment. For a high-level understanding of the CDE development environment, read *CDE Programmer's Overview*.

Features Exclusive to CDE

This section briefly describes the features of the CDE development environment that do not appear in the OpenWindows environment in any analogous form.

- Desktop Korn shell: Provides a way to engage in graphic user interaction through shell scripts. See *CDE Desktop KornShell User's Guide* for detailed information on desktop Korn shell.
- Workspace Manager: Provides support for multiple workspaces. Each workspace is a *virtual screen*. Most desktop applications can run as expected without knowledge of the Workspace Manager. See *CDE Programmer's Overview* and *CDE Programmer's Guide* for more information on the Workspace Manager.

OpenWindows Versus Solaris CDE Development Environments

This section contrasts some of the functionality that OpenWindows and Solaris CDE development environments have in common, but that might be implemented differently.

GUI Application Builders

Both the OpenWindows and CDE development environments contain GUI application builders.

- The OpenWindows Developer's Guide (Devguide) is a tool that helps produce the GUI for OPEN LOOK applications. Devguide has code generators that produced XView and OLIT source code. The Devguide Motif Conversion Utilities product, which was first shipped with the Solaris 2.3 release, produces Motif code from Devguide GIL files.
- CDE Application Builder (App Builder) produces CDE Motif code. It also enables you to integrate some of the desktop services into your application; for example, drag and drop, ToolTalk messaging, sessioning, help, and internationalization. If you used Devguide to create your application, you can use App Builder's GIL-to-BIL converter to create BIL files, which is the format the App Builder uses.

If you have used Devguide to build an application, the CDE Application Builder will feel very familiar to you. The palette is visually similar. Much of the Devguide functionality is retained, such as the Build and Test feature and the ability to build projects.

See *CDE Programmer's Overview* and *CDE Application Builder User's Guide* for more information on App Builder.

Drag and Drop

The underlying basic functionality of OpenWindows drag and drop and CDE drag and drop are similar. Both versions of drag and drop contain general purpose APIs that provide the same outcome from a user's perspective.

However, CDE drag and drop also provides a *convenience* API that serves two purposes:

- It is easier to use than the APIs that Motif 1.2.3 provides
- It defines and implements data transfer policies

The OpenWindows environment has conventions for drag and drop policies, but it is up to the developer to implement the policies. If you use drag and drop in your OPEN LOOK application, much of the data transfer code you wrote can be condensed by using the CDE convenience APIs.

See *CDE Programmer's Overview* and *CDE Programmer's Guide* for more information on CDE drag and drop.

ToolTalk Messaging

OpenWindows ToolTalk messaging is upwardly compatible with the CDE ToolTalk Messaging Service. In addition, the ToolTalk Messaging Service provides:

- Two standard ToolTalk protocols known as *message sets*
- `ttsnoop`

  The `ttsnoop` utility is a debugging tool that helps monitor messages that are sent between your application and other applications.

- `tttrace`

  The `tttrace` utility is a debugging tool that helps monitor ToolTalk API calls.

See *CDE Programmer's Overview*, *CDE ToolTalk Messaging Overview*, and the `ttsnoop(1)` and `tttrace(1)` man pages for more information on the CDE extensions of ToolTalk.

4

Typing

The OpenWindows *classing engine* identifies the characteristics, or attributes, of files. The classing engine specifies attributes such as print method, icons, and open commands for specific file types. The classing engine consists of two parts:

- A database that stores file type names and attributes
- A collection of routines that query the database

The Solaris CDE *data typing* and *actions* form the analog of the classing engine. The data-typing mechanism consists of two tables (DATA_ATTRIBUTES and DATA_CRITERIA) that specify attributes such as icons, actions, and commands for specific file types. DATA_CRITERIA corresponds to the classing engine's File name space. DATA_ATTRIBUTES corresponds to the classing engine's Type name space. The actions field in the DATA_ATTRIBUTES table corresponds to the former print and open methods in the classing engine. In the Solaris CDE development environment, it acts as a reference to another table called ACTION, and is greatly enhanced over the former classing engine methods.

See *CDE Programmer's Overview* and *CDE Programmer's Guide* for more information on data typing and actions.

Help

CDE help differs from OpenWindows help in three areas:

- User model
- Programming tasks
- Richness of help system

See *CDE Programmer's Overview* and *CDE Help System Author's and Programmer's Guide* for more information on the CDE help system.

User Model

The OpenWindows user model is very simple. The user places the pointer over the portion of the application he wants help on, and presses the Help key.

The CDE user model is similar but slightly different. The user receives help from that portion of the application that has input focus. That is, the user must select a portion of the application and then press the Help key to receive help on that portion.

CDE help provides *on item* help for those objects that cannot get input focus. You must provide an On Item selection in your application's Help menu so users can access help for those items.

Programming Tasks

Writing OLIT and XView help is very similar. In either case, your application provides the text appropriate for a widget or object, respectively. The OpenWindows window manager determines which application is responsible when the user presses the Help key, and dispatches an event to that application.

OLIT has a specialized Help widget, and XView has a Help frame, that puts the right information into the Help dialog display area and makes the dialog appear on the screen. You either have to include a function call to `OLRegisterHelp()` in OLIT, or set an object attribute in XView, to enable help on a widget (object).

To provide CDE help with your application, you must:

1. Establish help callbacks on all relevant widgets.

   These callbacks must be able to provide the help information for the associated widgets.

2. Create and manage the help dialogs.

   CDE provides `DtHelpDialog` and `DtHelpQuickDialog` widgets to create help dialog boxes and quick help dialogs. CDE has one shared help dialog that displays help.

3. Implement on item help.

   CDE Help System Author's and Programmer's Guide provides source code to do this.

Richness of Help System

OpenWindows help has a single help dialog in which you can place text only. CDE help is much richer. CDE provides both quick and full help. You can also include much more than just text in your help dialogs; for example, color graphics and hyperlinks. The user can print help and also use navigation facilities to stack and traverse help attached to different widgets.

Internationalization

You can internationalize an OpenWindows, as well as a CDE, application. The steps to internationalize an OpenWindows application that are different in CDE than in the OpenWindows environment are documented in Appendix B, "Internationalization and CDE."

See *CDE Programmer's Overview* and *CDE Internationalization Programmer's Guide* for more information on internationalizing a CDE application.

Fonts

In the OpenWindows environment, the default font type is Lucida. The same is true for the CDE desktop, so the CDE user will see no font style differences from the OpenWindows desktop.

Fonts in the OpenWindows environment have proprietary font names, whereas CDE is open. All CDE font names start with `-dt-` and are common across all CDE platforms. Both CDE and OpenWindows fonts follow the X11 XLFD font specifications.

In both the OpenWindows and CDE environments, your application should not override the system font defaults. This enables the user to customize his desktop. In the OpenWindows environment, the user can choose font type and style through the Workspace Properties desktop application. In CDE, the user can choose font size through the Style Manager.

In the OpenWindows environment, you have to specify fonts for any locale in which you want your application to run. By using the CDE Interface fonts, your application will run the same across all locales and desktops. If you use Application fonts, you must still specify fonts for Asian locales.

The CDE font aliases are in `/usr/dt/config/xfonts/<locale>`, where `<locale>` is the directory corresponding to the locale in which you are interested. The default font resources that the Style Manager uses are in `/usr/dt/config/<locale>/sys.fonts`.

See *CDE Programmer's Overview* and *CDE Programmer's Guide* for more information on CDE fonts.

4