Solaris Common Desktop Environment: Motif Transition Guide

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900
U.S.A.

Part Number 806-2917–10
February 2000

Contents

Preface

Solaris Common Desktop Environment: Motif Transition Guide addresses:

- Issues of concern to Sun Motif developers
- How to run existing OPEN LOOK and Motif applications on the OpenWindows™ 3.6 and Solaris Common Desktop Environment (CDE) desktops
- Porting OPEN LOOK and Motif applications to the Solaris CDE environment

This manual assumes you are familiar with OPEN LOOK or Motif programming. Use it in conjunction with Motif and OPEN LOOK manuals to enable your application to run on the latest Sun desktops.

Note - This book generically uses the terminology Solaris Motif for the Motif toolkit that is included with either the Solaris CDE unbundled software or the Solaris software.

Who Should Use This Book

Read *Solaris Common Desktop Environment: Motif Transition Guide* if you are:

- A Motif programmer interested in developing Solaris Motif applications for the OpenWindows 3.6 or Solaris CDE desktop
- An OPEN LOOK or Motif programmer, and you want your existing applications to run on the OpenWindows 3.6 or Solaris CDE desktop with little or no code modification
- Interested in porting your OPEN LOOK or Motif application to the Solaris CDE desktop

This manual assumes that you are proficient in OPEN LOOK (XView™ or OLIT) or Motif application development on UNIX® platforms. If you are an OPEN LOOK developer, it assumes you are familiar with Motif, as well.

Before You Read This Book

If you are considering porting your application to the Solaris CDE desktop, and you are not familiar with CDE, you should first read:

- *Common Desktop Environment: Programmer's Overview*

  Common Desktop Environment: Programmer's Overview provides a high level view of the CDE development environment components. It also includes an architectural overview of the entire CDE system, including both the run-time (end user) and development environments.

- *Solaris Common Desktop Environment: User's Guide*

  Solaris Common Desktop Environment: User's Guide provides an in-depth description of the CDE run-time environment.

See Appendix C for a listing of all the CDE documentation.

How This Book Is Organized

This manual consists of these chapters and appendixes:

Chapter 1 provides a roadmap to using this manual, depending on what types of tasks you want to perform on your application.

Chapter 2 contains information for developers writing Solaris Motif applications for either the OpenWindows or CDE environment.

Chapter 3 describes the Solaris Motif toolkit and identifies the non-standard parts of the Solaris 2.3 (IXI) Motif toolkit.

Chapter 4 compares and contrasts the OpenWindows and CDE development environments.

Chapter 5 discusses transitioning your application from an OPEN LOOK graphical user interface (GUI) to Solaris Motif.

Chapter 6 provides information to consider for porting your OPEN LOOK application to CDE.

Chapter 7 presents a simple porting example.

Appendix A describes the user interaction changes from the OPEN LOOK to the CDE user model.

Appendix B describes the things you must do differently from the OpenWindows environment to internationalize an application for the CDE desktop.

Appendix C lists books and articles on issues related to OPEN LOOK, Motif, and CDE application development.

Related Books

For a list of the CDE documentation and reading material of interest to OPEN LOOK and Motif developers, see Appendix C.

Ordering Sun Documents

Fatbrain.com, an Internet professional bookstore, stocks select product documentation from Sun Microsystems, Inc.

For a list of documents and how to order them, visit the Sun Documentation Center on Fatbrain.com at `http://www1.fatbrain.com/documentation/sun`.

Accessing Sun Documentation Online

The docs.sun.com[SM] Web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is `http://docs.sun.com`.

What Typographic Conventions Mean

The following table describes the typographic changes used in this book.

TABLE P–1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls –a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`machine_name%` **`su`** `Password:`
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type **`rm`** *filename.*
AaBbCc123	Book titles, new words, or terms, or words to be emphasized.	Read Chapter 6 in *User's Guide.* These are called *class* options. You must be *root* to do this.

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–2 Shell Prompts

Shell	Prompt
C shell prompt	`machine_name%`
C shell superuser prompt	`machine_name#`
Bourne shell and Korn shell prompt	`$`
Bourne shell and Korn shell superuser prompt	`#`

CHAPTER 1

Moving to Motif and CDE

This chapter discusses the different types of Motif available to Solaris CDE developers, and provides the paths through this manual for the different tasks you may want to perform in relation to them.

Summary of Motif Toolkits

This section summarizes the differences between the Motif toolkits supported by the Solaris environment. Any Solaris Motif application that is compiled on the Solaris 2.4, 2.5, 2.5.1, or 2.6 operating environment is binary compatible with Solaris 7 CDE.

- Solaris 7 Motif: Based on The Open Group's Motif 2.1 with additional bugfixes and enhancements. The Solaris 7 software always includes the Motif toolkit.
- Solaris 2.6 Motif: Based on OSF/Motif 1.2.5 with additional bugfixes and enhancements. The Solaris 2.6 software always includes the Motif toolkit.
- Solaris 2.5.1 Motif: Based on OSF/Motif 1.2.3 with additional bugfixes and enhancements. The Solaris 2.5.1 software always includes the Motif toolkit.
- Solaris 2.5 Motif: Based on OSF/Motif 1.2.3 with additional bugfixes and enhancements. The Solaris 2.5 software always includes the Motif toolkit.
- Solaris 2.4 Motif: Based on OSF/Motif 1.2.3 with additional bugfixes and enhancements. The exact release of Motif depends on what version of CDE has been installed on that system. The Solaris 2.4 software does not always include the Motif toolkit.

- Motif available for Solaris 2.3: A version of IXI Motif based on OSF/Motif 1.2.2. An application compiled with this Motif library may not run on later versions of Solaris. However any Solaris 2.3 Motif application that is currently running on Solaris 2.4, 2.5, or 2.5.1 will run on Solaris 2.6 in the same way. The Solaris 2.3 software does not always include the Motif toolkit.

Note - In the Solaris CDE environment, you gain access to additional widgets (such as a menu button widget and a terminal emulator widget and library) through the `libDtWidget` library. See Chapter 3 for more details.

Moving to Motif

Programming or porting tasks you probably want to perform in relation to Motif are:

- Running existing Motif applications on the OpenWindows 3.6 desktop
- Developing Solaris Motif applications for the OpenWindows 3.6 desktop

Running Existing Motif Applications on the OpenWindows Desktop

Any standard Motif 1.2 or Solaris Motif application will run on the OpenWindows 3.6 desktop.

Developing Solaris Motif Applications for the OpenWindows Desktop

If you are familiar with Motif and want to develop a Solaris Motif application for the OpenWindows 3.6 desktop, read Chapter 2.

You might also want to read Chapter 3, which compares the Motif toolkits that the Solaris environment supports.

If you are an OPEN LOOK developer and want to develop a Solaris Motif application for the OpenWindows 3.6 desktop, you should read the chapters and appendixes just mentioned in this section. In addition, read:

- Chapter 5, for a comparison of the OPEN LOOK and Motif graphical user interface (GUI) and widgets.
- Chapter 6, particularly those parts pertaining to GUI transitions.

- “Motif 2.1 Documentation” on page 84 and “Motif Programming” on page 86 for lists of books that will familiarize you with Motif programming.

Moving to Solaris CDE

You do not need to port your OPEN LOOK or Motif application to the Solaris CDE environment for it to run on the Solaris CDE desktop. However, if you decide to port, refer to the CDE documentation and read the chapters mentioned below. See “CDE Documentation” on page 83 for a list of the available CDE documentation.

The main programming or porting tasks you probably want to perform in relation to Solaris CDE are:

- Running existing OLIT, XView, or Motif applications on the Solaris CDE desktop
- Porting OpenWindows applications to the Solaris CDE desktop
- Porting Motif applications to the Solaris CDE desktop
- Developing Solaris Motif applications for the Solaris CDE desktop

Running Existing Applications on the Solaris CDE Desktop

Existing OLIT, XView, or Motif applications will run on the Solaris CDE desktop.

If you want your application to appear integrated with the Solaris CDE desktop but do not want to modify your application code, you can perform basic integration. This first level of Solaris CDE integration is described in “Basic Integration” on page 50.

Porting OpenWindows Applications to the Solaris CDE Desktop

Existing OpenWindows applications can run unmodified on the Solaris CDE desktop. If you want to move your applications to the Solaris CDE environment and to begin using its broader set of standard services, read this book to help you understand the differences between the OPEN LOOK and Motif toolkits, and the OpenWindows and Solaris CDE desktops.

Porting Motif Applications to the Solaris CDE Desktop

If your application is Motif 2.1 style guide-compliant, you are well on your way to it being CDE style guide-compliant. Solaris Motif is based on the Motif 2.1 toolkit and the CDE style guide is based on the Motif 2.1 style guide. Still, you may have to make some GUI changes to port your application to Solaris Motif.

To help port your Motif application to the Solaris CDE desktop, read:

- Chapter 3, to find out about the enhancements in Solaris Motif and the CDE widgets available to you
- Chapter 4, which compares and contrasts the OpenWindows and Solaris CDE development environments

Refer to the checklist in the *Common Desktop Environment: Style Guide and Certification Checklist* to see how the CDE style guidelines are similar to and differ from those for Motif 1.2.

Developing and Debuggging Applications for the Solaris CDE Desktop

Solaris CDE shared libraries are built with the latest Solaris loader technology to optimize their interfaces and performance. This technology conflicts with debuggers that were released prior to SPARCworks version 3.0.1. Therefore, use SPARCworks version 3.0.1 or later when developing and debugging applications in CDE.

Developing Solaris Motif Applications

If you are familiar with Motif and want to develop a Solaris Motif application for the Solaris CDE desktop, refer to the CDE documentation.

If you are an OPEN LOOK developer and want to develop a Solaris Motif application for the Solaris CDE desktop, read this book and refer to the CDE documentation.

In either case, see Appendix C for a list of the CDE documentation as well as other books to help you with Motif programming.

Solaris CDE Naming Conventions

Solaris CDE uses the prefixes *DT* and *SDT* in uppercase and lowercase combinations, in names for desktop clients, desktop libraries, and so on. Do not use these prefixes in any Solaris CDE desktop application you write.

CHAPTER 2

Motif Environment

This chapter contains information for developers writing Solaris Motif applications for either the OpenWindows or Solaris CDE environment.

Motif Packaging in the Solaris Software

The Motif run-time support in the Solaris 2.6 software includes the following:

- Shared libraries
- Header files
- Key bindings
- `uil` compiler
- Man pages
- Demos and sample source

The Motif header files required for application development are located in `/usr/dt/include`. The Motif libraries are located in `/usr/dt/lib`.

Compiling and Linking Motif Programs

When you compile Motif programs, include the following compiler syntax to enable the compiler to find the Motif and X Window System™ header files:

```
-I/usr/dt/include -I/usr/openwin/include
```

Use the following compiler syntax to direct the linker to the correct shared libraries as shown in the following:

```
-R/usr/dt/lib -R/usr/openwin/lib -L/usr/dt/lib -L/usr/openwin/lib
```

The following is an example of a compile-and-link line for a Motif application:

```
cc -o myprog -I/usr/dt/include -I/usr/openwin/include myprog.c -R/usr/dt/lib \
-R/usr/openwin/lib -L/usr/dt/lib -lXm -L/usr/openwin/lib -lXt -lX11
```

Shared Library Policy

Sun will increment the major version number of each shared Motif library whenever there are binary-incompatible differences from the previous release. Sun will make available (either on the Motif distribution or through some other channel) all prior versions of each library. This will ensure that your applications linked with a particular release can continue to run, even after a new Motif release has been installed.

CHAPTER 3

Solaris Motif Toolkits

This chapter provides some information about Solaris 2.6 Motif and IXI Motif, and introduces the widgets available for Solaris CDE application development.

Motif Toolkits in Solaris Software

This section discusses the Solaris 2.6 Motif and IXI Motif toolkits. For a summary of the differences between the Motif toolkits available for Solaris application development, see "Summary of Motif Toolkits" on page 13.

Solaris 7 Motif

Solaris 7 Motif: Based on The Open Group's Motif 2.1 with additional bugfixes and enhancements. The Solaris 7 software always includes the Motif toolkit.

Enhancements to Existing Motif 2.1 Functionality

The Solaris Motif library contains minor enhancements to Motif 2.1 usability to emulate certain OPEN LOOK user interface and Microsoft® Windows features. The usability enhancements include:

- Optionally allowing mouse button 2 on a three-button mouse to be used to extend the current selection. This is equivalent to the OPEN LOOK Adjust function.

- Allowing mouse button 3 to activate a CascadeButton menu (for OPEN LOOK compatibility).
- Ability to re-map keybindings to be consistent with those for OPEN LOOK or Microsoft Windows applications.

Solaris CDE Libraries for Motif

Motif Library (libXm)

Solaris CDE provides all the Motif 1.2.5 header files. The Solaris CDE libraries for Motif are the Motif 1.2.5 libraries with bug fixes and enhancements.

Motif UIL library (libUill)

The Motif user interface language (UIL) is a specification language for describing the initial state of a Motif application's user interface. The CDE version of the Motif UIL library is essentially unchanged from the Motif 1.2.5 version.

Include the `UilDef.h` header file (found in the `/usr/dt/include/uil` directory) to access UIL.

Motif Resource Manager Library (libMrm)

The Motif resource manager (MRM) is responsible for creating widgets based on definitions contained in user interface definition (UID) files created by the UIL compiler. MRM interprets the output of the UIL compiler and generates the appropriate argument lists for widget creation functions. Use `libMrm` to access the Motif resource manager. The CDE version is essentially unchanged from the Motif version.

Include the `Mrm/MrmPublic.h` header files to access `libMrm` in your application.

Related Documentation

See the *OSF/Motif Programmer's Reference* for information on UIL, the UIL compiler, UID, and Mrm.

IXI Motif 1.2.2 Toolkit

The IXI Motif 1.2.2 toolkit, which was available for Solaris 2.3 software development, contains some incompatibilities with standard OSF/Motif 1.2.2. These features are

not part of the OSF/Motif 1.2 specification, and are not present in the Solaris 2.4 and later Motif toolkits.

XmList convenience functions

The following are the non-standard functions in IXI Motif. Remove them from your application code if you have used them:

- `XmListRecolorItem()`
- `XmListRecolorPos()`
- `XmListSetClientDataPos()`
- `XmListSetClientDatasPos()`

XmForm widget

The `XmForm` widget implementation in IXI Motif and OSF/Motif 1.2.2 are different, although the APIs are identical. Hence, applications linked with IXI Motif can exhibit minor behavioral or visual differences in the `XmForm` widget when they are re-linked with OSF/Motif.

The Solaris Motif toolkits use the OSF/Motif `XmForm` widget implementation, which is binary compatible with the OSF/Motif 1.2.2, but not the IXI Motif, `XmForm` widget.

Complex Text Layout (CTL) Support

Solaris 7 software supports the five new CTL widgets introduced by Motif 2.1. This is achieved by a single binary developed on the Solaris 7 operating environment that provides advanced and standard support for Hebrew, Arabic and Thai customers.

The following new Motif widgets are supported.

- XmNotebook is a full featured widget that provides functionality similar to a notebook or “tab” widget
- XmContainer is a full featured GUI icon “tree” dsplay widget
- XmSpinBox is a user control to increase and decrease a numerical text field
- XmScale widget has changed to provide a new vertical display

Widgets Available for CDE Application Development

This section discusses the widgets available for Solaris CDE application development, as an extension to Solaris Motif.

Solaris Motif Control Widgets

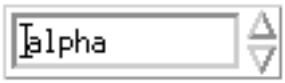
The Solaris Motif control widgets are designed to ease porting OPEN LOOK applications to the Solaris CDE desktop by providing equivalent functionality in Solaris Motif. These widgets are not considered to be part of Solaris Motif, but rather an extension to Solaris Motif. The `libDtWidget` library contains widgets and functions that are used to provide common functionality across all CDE applications. The widgets provided include:

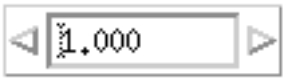
TABLE 3–1 CDE Control Widgets

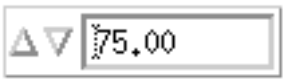
Widget Name	Description
`DtSpinBox`	`TextField` widget with additional controls for incrementing and decrementing numeric values, or browsing through and selecting from a list of text strings. Can be read-only. `DtSpinBox` is functionally similar to the OPEN LOOK numeric text field.
`DtComboBox`	Combination of `TextField` and pop-up list widgets that provides a list of valid choices for the `TextField`. Can be read-only.
`DtMenuButton`	Command widget that provides the menu cascading functionality of an `XmCascadeButton` widget outside of a menu bar, or a menu pane. `DtMenuButton` is functionally equivalent to the OPEN LOOK menu button.

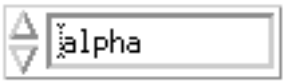
Examples of each type of widget follow:

- Text field and arrow button widget (`DtSpinBox`)

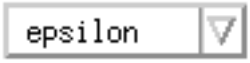




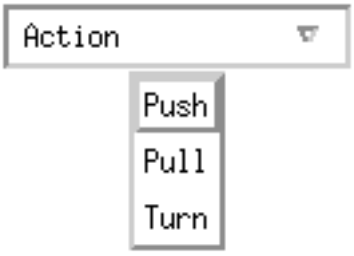




- Text field and list box widget (`DtComboBox`)



- Menu button widget (`DtMenuButton`)



The `DtComboBox` custom widget calls the Selection Callback of the Combo Box widget, instead of the List widget, when the `browseSelectCallback` or the `defaultActionCallback` is triggered for the List widget of a Combo Box.

This action may cause failure for applications that rely on the List widget Selection callback to be called when the List widget's `browseSelectCallback` or `defaultActionCallback` is triggered.

Note - The Solaris CDE software supports all Motif 1.2 widgets.

Compatibility with Motif 2.1

The APIs of the `DtSpinBox` and `DtComboBox` widgets are similar to the Motif 2.1 release of `XmSpinBox` and `XmComboBox` widgets. The APIs are designed so an application can easily switch to the Motif 2.1 version of these widgets. The main

thing you need to do to switch is to change the `Dt` names for the class, types, and creation routines to `Xm`.

This information is supplied in case you choose to port your application to Motif 2.1 but this is *not* a recommendation that you do so.

Note - The Solaris CDE software does *not* guarantee strict API or binary compatibility between its widgets and the Motif 2.1 widgets.

Library and Header Files

The library `libDtWidget` provides access to the `DtSpinBox`, `DtComboBox`, and `DtMenuButton` widgets. The `libDtWidget` header files for these widgets are:

- `Dt/SpinBox.h`
- `Dt/ComboBox.h`
- `Dt/MenuButton.h`

Demo Programs

You can find the Solaris Motif control widgets demos in `/usr/dt/examples/dtwidget`. Read the `README` file for detailed information on the demos.

Related Documentation

For more information on Solaris Motif control widgets, see the relevant man pages and *Solaris Common Desktop Environment: Programmer's Guide*.

CDE Terminal Widget

The `DtTerm` widget is part of the Solaris CDE development environment. It provides the functionality required to emulate an ANSI X3.64-1979-style terminal (specifically a DEC® VT220-like terminal with extensions). The Terminal widget library, `libDtTerm`, provides the `DtTerm` widget for adding a terminal window to a GUI. If you include a terminal in your application, use Solaris Motif widgets to add display enhancements to it such as pop-up menus and scrollbars.

The Solaris CDE Terminal application, which is a part of the run-time environment, is a window that behaves as a terminal, enabling access to traditional terminal-based applications from within the desktop. The `DtTerm` widget is the foundation for the desktop run-time terminal.

The libDtTerm library includes a set of convenience functions to create, access, and support the DtTerm widget.

Library and Header Files

The libDtTerm library provides a set of widgets based on Motif for designing a terminal or for adding a terminal window to a GUI.

Include the Dt/Term.h header file to access libDtTerm APIs in your application.

Demo Programs

You can find the DtTerm demos in /usr/dt/examples/dtterm. See the README file for detailed information on the demos.

Related Documentation

For more information on the DtTerm widget, see the relevant man pages.

For more information on the desktop terminal application, see the terminal help volume, the relevant man pages, or *Solaris Common Desktop Environment: User's Guide*.

Text Editor Widget

The CDE text editing system consists of two components:

- The Text Editor application, which provides editing services through graphical, action, and ToolTalk interfaces
- The editor widget, DtEditor, which provides a programmatic interface for the following editing services:
  - Cut and paste
  - Search and replace
  - Simple formatting
  - Spell checking (for 8-bit locales)
  - Undo previous edit
  - Enhanced I/O handling capabilities that support input and output of ASCII text, multibyte text, and buffers of data
  - Support for reading and writing files directly

Although the Motif text widget also provides a programmatic interface, applications that want to assure a system-wide uniform editor should use the DtEditor widget.

The CDE Text Editor and Mailer applications use the editor widget. Use this widget in the following circumstances:

- You need the functionality, such as spell checking, undo, and find/change, that is provided by the `DtEditor` widget.
- You want users to be able to read and write data to and from a file.
- When your program does not need to edit the text while the widget has control of the text.

Library and Header Files

The `DtEditor` widget is in the `libDtWidget` library. The header file is `Dt/Editor.h`.

Demo Programs

A demo containing an example of the `DtEditor` widget (`editor.c`) is in `/usr/dt/examples/dtwidget` directory. Read the `README` file for detailed information on the demo.

Related Documentation

For more information on the Text Editor widget, see the relevant man pages and *Solaris Common Desktop Environment: Programmer's Guide.*

CHAPTER 4

Development Environment Transition Issues

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.

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.

Features Exclusive to Solaris CDE

This section briefly describes the features of the Solaris 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 *Common Desktop Environment: Desktop KornShell User's Guide* for detailed information on desktop Kornshell.
- 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 *Common Desktop Environment:*

Programmer's Overview and *Solaris Common Desktop Environment: 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 Solaris 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 Solaris 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 *Common Desktop Environment: Programmer's Overview* and *Common Desktop Environment: 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 *Common Desktop Environment: Programmer's Overview* and *Solaris Common Desktop Environment: Programmer's Guide* for more information on CDE drag and drop.

ToolTalk Messaging

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

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

  The `tttrace` utility is a debugging tool that helps monitor ToolTalk API calls and ToolTalk's internal message processing.
- `ttsnoop`

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

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

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 *Common Desktop Environment: Programmer's Overview* and *Solaris Common Desktop Environment: 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 *Common Desktop Environment: Programmer's Overview* and *Common Desktop Environment: 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.**

   Common Desktop Environment: 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 .

See *Common Desktop Environment: Programmer's Overview* and *Common Desktop Environment: 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 *Common Desktop Environment: Programmer's Overview* and *Solaris Common Desktop Environment: Programmer's Guide* for more information on CDE fonts.

Motif Applications and Color

The CDE environment changes the color threshold values for Motif; hence, Motif applications may have different colors under Solaris CDE from those in the OpenWindows environment.

CHAPTER 5

Toolkit Transition Issues

This chapter describes toolkit terminology, common widgets, CDE exclusive widgets, and high-level widgets, and provides information about the different libraries.

OPEN LOOK Versus Solaris Motif Toolkits

When you compare an OPEN LOOK application with its Solaris Motif counterpart, a few contrasting visual elements are immediately apparent. For example, the OPEN LOOK buttons are round whereas the Solaris Motif buttons are square. The shading applied to buttons and other objects for a three-dimensional appearance are also different. Although such cosmetic elements do not affect a program's behavior, and can often be disregarded when porting, your application will not be CDE style-guide compliant if you deviate from the CDE look.

Several differences are often significant in a conversion effort. The most critical of these features and other GUI elements are summarized in three sections:

- Aspects of the OPEN LOOK user interface that are missing from Solaris Motif
- Aspects of Solaris Motif that do not appear in the OPEN LOOK user interface
- Features or other elements that appear in both specifications but are implemented differently

See the Preface for a list of style guides and other references that describe the OPEN LOOK and Solaris Motif GUIs.

See Appendix A for a more detailed list of the user interaction changes from OPEN LOOK to CDE.

Features Only in the OPEN LOOK User Interface

The following features are found in the OPEN LOOK user interface and are implemented in XView or OLIT (or both) but do not appear in Solaris Motif:

- Split window control
- Scrollbar anchors
- Defaults of the menu
  - Automatic default; shortcut method for selecting the default
  - Default menu item indicated by a *ring*
- Font chooser widget
- Pointer warping in notices
- Can drop onto minimized windows (icons)

Features Only in the Solaris Motif User Interface

The following features are found in Solaris Motif and not in the OPEN LOOK user interface:

- Front Panel
- Tools for producing and registering help volumes with hypertext capabilities
- Additional user interface objects (*widgets*) available for applications and specified in the CDE Style Guide

Notable Implementation Differences Between Toolkits

Many features are roughly equivalent in the OPEN LOOK user interface and Solaris Motif but have significant implementation differences. The following are the most important differences:

- Tear off Menus
- Input focus indicators

- Widget classes (sliders and gauges versus scales, for example)
- Secondary text selections

Other significant differences include the following:

- Keyboard bindings
- The window manager controls associated with each window
- Internalization
- Mouse button behavior

The remainder of this section provides information on some of these features.

Tear-off Menus

In Solaris Motif, tear-off menus replace the *pinned menus* of the OPEN LOOK user interface. Selecting the dashed line on the top of the menu “tears-off” the menu.

Secondary Text Selection

The OPEN LOOK secondary text selection is roughly analogous to the Motif quick transfer mechanism.

Window Controls

One of the most critical implementation differences involves window controls. When the Motif user presses the Window menu button in the upper left corner of the title bar, or the OPEN LOOK user presses the MENU button anywhere on the window background (including the header), a menu is displayed. The options offered under the two GUIs introduce a key contrast.

The Motif *Window menu* offers a choice of Restore, Move, Size, Minimize, Maximize, Lower, and Close. The OPEN LOOK *base Window menu* offers Close, Full Size, Properties, Back, Refresh, and Quit. The two lists are fundamentally the same, but have very different effects.

In the OPEN LOOK user interface, the Close option minimizes (iconifies) the window, and Quit terminates the application.

In Motif, the Minimize option minimizes the window, and the Close option terminates the application. Many users familiar with the OPEN LOOK user interface have found themselves exiting a Motif program when their intent was to close its window to an icon.

Mouse Button Behavior

The structure of the mouse buttons is very similar in both specifications; however, the difference is significant enough to cause some confusion.

Table 5–1 shows the default left-to-right mapping of the three OPEN LOOK mouse buttons.

TABLE 5–1 OPEN LOOK Mouse Buttons

BUTTON	Description
SELECT	Specifies an object or manipulates objects and controls, drag
ADJUST	Extends or reduces the number of selected objects
MENU	Displays a menu associated with the pointer location or specified object

The three Motif mouse button assignments, described in Table 5–2 , also start by default with the left mouse button.

TABLE 5–2 Motif Mouse Buttons

Button	Description
BSelect	Selects, activates, and sets the location cursor, drag
BTransfer	Moves and copies elements, drag and drop transfer. Can be customized to be the OPEN LOOK Adjust button.
BMenu	Displays menus

XView Libraries Versus Solaris Motif Libraries

This section compares the XView and Solaris Motif libraries.

Terminology

The XView toolkit and the Xt (OLIT and Motif) toolkits use the following terminology:

XView	Xt (OLIT and Motif)
Package	Widget
Attribute	Resources
Frame	Shell

XView is based directly on Xlib, whereas Motif is based on the Intrinsics (Xt) toolkit and the Intrinsics toolkit is based on Xlib.

XView	OLIT	Motif
XLib	Xt	Xt
	XLib	XLib

Because of this fundamental difference, the basic library functions to initialize the environment and create, modify and destroy graphical objects, are different, as shown in the following examples:

XView	Motif/Xt
`xv_init()`	`XtAppInitialize()`
`xv_create()`	`XtCreateWidget()`
`xv_set()`	`XtSetValues()`
`xv_destroy()`	`XtDestroyWidget()`

Functions that deal with event handling callbacks, and internationalization features, for example, get more complicated. For these features, simple one-to-one correspondence does not exist.

OPEN LOOK Versus Motif Toolkit Architecture

OLIT and Motif have very similar architectures whereas XView and Motif do not. When migrating from XView to Motif, you should note these toolkit differences.

The XView toolkit implements both the user interface objects, called *packages*, and the routines and processes that hold the interface together (creation routines, event processing, and so on), while Motif and OLIT implement basically just the user interface object, *widgets*, leaving the routines and processes to the Intrinsics library. For example, `xv_init( )` is a function that is in the XView library. The corresponding function to use for OLIT or Motif programming, `XtAppInitialize( )`, is in the Xt Intrinsics library.

The three toolkits represent two different GUIs. The appearance of the XView and OLIT toolkits are similar, whereas the appearance of Solaris Motif is noticeably different. Although there is a rough one-to-one correspondence between the function calls in the libraries, the behavior of parallel programs is different. That is, even after an OLIT (or XView) program has been converted to use the Solaris Motif library, it still to some degree has an OPEN LOOK appearance. A program in such a state does not adhere to either style completely.

Programming Model

Although the APIs are different, both XView and Motif are based on the same object-oriented methodology for programming a user interface:

- Initialize the toolkit
- Instantiate user interface objects
- Register callbacks on the user interface objects
- Enter event loop, waiting for user to generate events on the user interface objects

The overall structure of a program being ported from XView to Motif can remain intact even though all the function calls must be converted from one API to the other.

Common Types of User Interface Objects

Both XView and Motif are user interface toolkits that support some common types of user interface objects. However, XView implements many of these objects at a higher level that requires more than one Motif widget to produce the equivalent of a single XView object.

Table 5–3 lists the basic mapping of common objects for XView and its equivalent Solaris Motif widget:

TABLE 5–3 Basic Mapping of Common Objects.

XView Package	Equivalent Solaris Motif Widget
Base Frame	`XmTopLevelShell` + `XmMainWindow`
Command Frame	`XmDialogShell` + `XmBulletinBoard`
Notice	`XmDialogShell` + `XmMessageBox` (MessageDialog)
Canvas	`XmScrolledWindow` + `XmDrawingArea`
Panel	`XmBulletinBoard` or `XmForm`
Panel Button	`XmPushButton`
Menu Button	`DtMenuButton` (not accessible in OpenWindows environment)
Abbrev Menu Button	`XmRowColumn` (Option Menu)
Checkbox	`XmRowColumn` + `XmToggleButtons` (CheckBox)
Exclusive Choice	`XmRowColumn` + `XmToggleButtons` (Radio Box)
Scrolling List	`XmScrolledWindow` + `XmList`
Message	`XmLabel`
Slider	`XmScale`
Text Field	`XmTextField`
Numeric Text Field	`DtSpinBox` (not accessible in OpenWindows environment)
Text SubWindow	`XmScrolledWindow` + `XmText` (ScrolledText)
TTY SubWindow	`XmScrolledWindow` + `XmTermPane`; `DtTerm` (not accessible in OpenWindows environment)
Scrollbar	`XmScrollBar`

TABLE 5–3 Basic Mapping of Common Objects. *(continued)*

XView Package	Equivalent Solaris Motif Widget
Popup Menu	`XmMenuShell` + `XmRowColumn` (Popup Menu)
Pulldown Menu	`XmMenuShell` + `XmRowColumn` (Pulldown Menu)
Pullright Menu	`XmMenuShell` + `XmRowColumn` (Pulldown Menu)
File Chooser	`XmDialogShell` + `XmFileSelectionBox`

Differences Between XView and Solaris Motif

XView abstracts a number of other X11 functions that Motif (Xt) does not. In order to get equivalent functionality in Motif, these must be re-coded with direct Xlib calls.

The XView packages with no Motif equivalents are:

- Icon
- Server Image
- Colormap Segment (CMS)
- Cursor
- Fullscreen
- Server
- Font

Additionally, *no* Solaris Motif equivalent exists for the OPEN LOOK user interface functionality that splits windows into different views. To implement this function, you must subclass one of the Motif Manager widgets.

Some XView functions can be coded with the Motif/Xt API; however, these APIs are significantly different, and require complete redesign and recoding. These functions are:

XView API	Motif/Xt
Selection service	Xt selection API
Drop target package	Motif drag-and-drop API[1]
Notifier	Xt event management API

1. If you are porting to CDE, use the CDE drag-and-drop API. It is an extension of the Motif 1.2 drag-and-drop API and is easier to use.

X Resources

The X resources that control the behavior and appearance of XView and Motif applications are different. XView objects may not necessarily have instance names attached to them, unlike Motif/Xt objects. XView resources are independent of instance names. For example, `Window.Foreground.Color` affects all relevant XView objects.

Motif/Xt resources contain either class or instance names, for example, `mainframe.control_panel.button1.foreground`. To make Xt resources affect more than one object, use wildcards and class names.

External Files

XView reads in several files at start-up time ranging from message files to application specific default files containing X resources. The content and location of some of these files are different from Motif. (For example, Motif/Xt does not read in a message domain file (`.mo` file) under `$OPENWINHOME`.) The internationalization messaging scheme is completely different for Motif.

OLIT Libraries Versus Solaris Motif Libraries

Solaris Motif and OLIT support a number of convenience routines that are useful in manipulating objects in the user interface.

Routines Only in OLIT Library

These features are unique to the OLIT library.

- Error-Handling Routines

  OLIT provides a range of routines to enable the application to customize error handling.

- Dynamic Resources

  OLIT provides support for the user to dynamically change the value of certain resources (such as colors or fonts) after an application has been invoked.

Routines Only in Solaris Motif Libraries

These features are unique to the Solaris Motif libraries.

- UIL Support

  Solaris Motif supports the engine for using UIL definitions for the user interface layout. This separation allows the user interface to be modified without recompiling the program's executables. OLIT's Devguide solution (`golit`) provides similar functionality by allowing the user interface to be defined in GIL file format; however, the application must be recompiled when the user interface changes.

- Clipboard Routines

  Solaris Motif provides a library for managing the clipboard and its selections.

- Widget-Creation Routines

  Solaris Motif provides a complete set of routines that create a particular type of widget or group of widgets.

- Compound String Support

  Solaris Motif requires the user of compound strings for most text. To support these special string formats, Solaris Motif includes a number of routines to create and manipulate compound strings.

Widgets

Both the OLIT and Solaris Motif toolkits support a number of common widgets and gadgets with similar functionality, and each supports a number of more exclusive widgets. If a widget is implemented in one toolkit but not in the other, you can often build an equivalent object using multiple widgets in the other toolkit.

Table 5–4 matches the common widget name to the actual class name of the widget in each toolkit.

TABLE 5–4 Common Widget Mapping

OLIT Class Name	Solaris Motif Class Name
BulletinBoard	XmBulletinBoard
DrawArea	XmDrawingArea
ExclusiveChoice + RectButtons	XmRowColumn + XmToggleButtons (Radio Box)
Form	XmForm
Manager	XmManager
MenuButton	DtMenuButton (not accessible in OpenWindows environment)
NoticeShell	XmDialogShell + XmMessageBox
AbbrevMenuButton	XmRowColumn (Option Menu)
PopupWindowShell	XmDialogShell
NonExclusiveRectButton	XmRowColumn + XmToggleButtons (Check Box)
PopupMenuShell	XmMenuShell
Primitive	XmPrimitive
OblongButton	XmPushButton
ControlArea	XmRowColumn
Scrollbar	XmScrollBar
ScrollingList	XmList + XmScrolledWindow
ScrolledWindow	XmScrolledWindow
Slider	XmScale

TABLE 5–4 Common Widget Mapping *(continued)*

StaticText	XmLabel
TextEdit	XmText
TextField	XmTextField
RectButton	XmToggleButton

Widgets Exclusive to Solaris Motif

This section briefly describes the widgets that are exclusive to Solaris Motif.

Note - These widgets are also available for Motif development in the OpenWindows environment, except for the `DtTerm`, `DtEditor`, `DtComboBox`, `DtSpinBox`, and `DtMenuButton` widgets.

- `DtTerm`

  This widget provides the functionality required to emulate an ANSI X3.64-1979-style terminal emulator (specifically a DEC VT220-like terminal with extensions).

- `DtEditor`

  This widget provides a programmatic interface for editing services such as cut and paste.

- `DtComboBox`

  This widget is a combination of a text field and a list widget that provides a list of valid choices for the text field. Selecting an item from this list automatically fills the text field with that list item.

- `DtSpinBox`

  This widget is a convenient user interface control that increments and decrements an arbitrary TextField.

- `DtMenuButton`

  This widget is a command widget that complements the menu cascading functionality of an `XmCascadeButton` widget. (OLIT has its own menu button widget, with equivalent functionality to DtMenuButton.)

- `XmArrowButton`

This button is a primitive push button widget that displays an arrow label.

- `XmCommand`

  This is a manager widget that builds a command box and manages the user-selected commands and command history.

- `XmDrawnButton`

  This button is a primitive push button whose label can be drawn by the program.

- `XmFrame`

  This manager widget is used to parent a single child and enclose that child with a frame or border.

- `XmLabelGadget`

  This gadget is a low-overhead object for read-only text.

- `XmMainWindow`

  This manager widget supports a menu bar, command area, and work area.

- `XmPanedWindow`

  This manager widget implements resizeable panes within a window.

- `XmSelectionBox`

  This widget box allows you to select one item from a list in a general dialog box.

- `FileSelectionBox`

  This widget provides a standard way to select a file (typically for the application to read or write).

The `libDtWidget` library, which contains the `DtComboBox`, `DtSpinBox`, `DtMenuButton` and `DtEditor` widgets, depends directly on the following libraries:

- Xm library for the Motif superclass support
- Xt library for creation and manipulation of widgets
- X library for the base X Window System

Widgets Exclusive to OLIT

This section briefly describes the widgets that are exclusive to OLIT.

- `DropTarget`

  This primitive widget implements both the source and destination ends of drag-and-drop operations.

- `FlatWidget`

  These special widgets manage any number of subobjects within the context of a single widget. When implementing menus or choice objects that contain many subitems, they provide a significant memory savings.

- `FooterPanel`

  This manager widget automatically supports a window with a floating footer area.
- `RubberTile`

  This manager widget allows relative-sizing constraints to be placed on its children.
- `Stub`

  This primitive widget enables you to customize its behavior without subclassing.

CHAPTER 6

Porting Issues and Ideas

This chapter discusses porting OPEN LOOK applications to Solaris Motif on the Solaris CDE desktop. Much of the information presented is generic enough to encompass porting from OPEN LOOK to Motif running on the OpenWindows desktop.

Elements of Migrating

Migrating from the OPEN LOOK user interface to Motif is complex. It generally will not amount to a widget-for-widget swap. Do not expect it to be as straightforward as a line-by-line code translation. Depending on your application, the migration can range from a major architectural impact down to subtle widget differences.

Besides migrating from the OPEN LOOK user interface to Motif, porting to the Solaris CDE desktop means that you have the Solaris CDE development environment infrastructure available to your application. See Chapter 4 for a summary of some of

these features. See *Common Desktop Environment: Programmer's Overview* for a more detailed description of the development environment components and documentation.

Do You Need to Port?

First you should decide whether you really have to port your application. As mentioned in "Running Existing Applications on the Solaris CDE Desktop" on page 15, OPEN LOOK and Motif applications run "as is" on the Solaris CDE desktop. So you do not *need* to port your existing applications to Motif or CDE to have them run on the Solaris CDE desktop.

This provides you with flexibility about when and under what circumstances you decide to port your application. For example, you may decide to wait until a major release of your product before porting your application to the Solaris CDE desktop.

Basic Integration

Basic application integration is a set of highly recommended tasks you should perform to integrate your application into the Solaris CDE desktop. These tasks *do not* require modification of the source code for your application. (Some types of print integration—enabling printing in your application—require slight code modification, but these are optional to basic integration.)

Basic integration does not involve extensive use of the desktop application program interface (API). Therefore, it does not provide other interaction with the desktop, such as drag-and-drop capabilities.

A lot can be done to integrate your application into the Solaris CDE desktop without modifying any code. You can:

- Define actions for your application
- Write a help volume and have it available from the top level of the Help Manager
- Integrate your application with the Front Panel and Application Manager
- Enable printing

The Solaris CDE desktop provides interoperability between your application and other desktop applications. You can add new services (see "Recommended Integration" on page 51 and "Optional Integration" on page 52) if you want to use them and are willing to modify your code.

See *Solaris Common Desktop Environment: Programmer's Guide* for details on how to enable printing in your application, and for a list of the steps that comprise basic integration. See the "Registering an Application" chapter of *Solaris Common Desktop*

Environment: Advanced User's and System Administrator's Guide for detailed instructions on how to implement the basic integration steps.

If You Decide to Port

If you decide that you definitely want to port, think of the process as an art and not as a science. There is no magic tool that will perform the port for you. There is no foolproof algorithm to follow that works every time. What you really need to do is learn Motif and CDE, and understand the CDE style guidelines. This takes time and patience.

Benefits of Porting

Benefits of porting to the CDE desktop include:

- Taking advantage of new features offered by CDE
- Providing easy portability on a wide variety of platforms
- Enabling better "ease of use" by ensuring that your application behaves according to the CDE style guidelines
- Providing interoperability with other Solaris CDE applications in areas where the Solaris CDE and OpenWindows environments differ, for example:
  - Window manager interaction
  - Drag and drop
  - Session management

Integrating into the Solaris CDE Environment

Recommended and optional integration require changes to your code to implement the functionality within these categories. Your application will be even more integrated with the desktop the more CDE functionality you adopt.

Recommended Integration

The Solaris CDE development environment contains components and guidelines so that your application will integrate well with other applications on the desktop:

- Help system
- ToolTalk messaging system

- Session Manager
- Drag and drop
- Internationalization
- Standard font names
- Error message guidelines
- User customization issues

Optional Integration

The following Solaris CDE components enable you to leverage services provided by the desktop for achieving specialized tasks:

- Solaris Motif control widgets
- Data typing
- Action invocation
- Workspace Manager
- Terminal widget
- Text editor widget
- Calendar API
- Desktop Korn shell

Start Small

If you plan to port your application to the Solaris CDE desktop, or if you want to practice porting, start with a small example and work your way up.

Convert and Clean Up

With small applications that have simple GUIs, you can practice porting by using a two-step process:

- Convert the GUI object-by-object from the OPEN LOOK user interface to Motif. (See “Use a Motif GUI Builder” on page 53 below.)
- Clean up the resulting GUI so that it adheres to the CDE style guidelines. Take advantage of this opportunity to review the GUI for ease-of-use and customer-specific issues. You might decide to change the interface, even if the object-by-object conversion is style-guide compliant.

Chapter 7, takes a simple OPEN LOOK application and illustrates this process.

Note - This is *not* the recommended way to port large, real-life applications.

Use a Motif GUI Builder

Use a Motif GUI builder such as Application Builder (App Builder) or SunSoft Visual WorkShop™ to build the new Motif GUI for the application. You need flexibility to experiment and make changes when you port using the two-step process described in “Convert and Clean Up” on page 52. If you manually code in a particular GUI, it is difficult and time consuming to make modifications. App Builder provides flexibility by enabling you to drag and drop objects to easily create prototype GUIs. It generally requires *less* time to use a GUI builder to lay out a new Motif interface than it does trying to port the GUI by hand.

In fact, whatever porting process you use and no matter how large your application, you should consider using a Motif GUI builder. These builders produce the GUI and application framework code, which frees your time to focus on application code.

Architectural Impact

Resist the urge to dive right in and start transforming your code to Solaris Motif. Begin the process of porting your application to Motif and CDE by examining your application’s architecture.

The more architecture your application has, the more important it is to re-architect the application correctly up front. In these situations the complexity of the porting process increases dramatically if you use the “convert and clean up” strategy.

GUI and Internals

For programs in which important functions are insulated from the surrounding GUI, the impact of the difference in the OPEN LOOK user interface and Motif can be negligible. However, if the code is tightly linked to the user’s actions or relies on a specific OPEN LOOK feature, producing a Solaris Motif equivalent may be difficult.

If you can draw a line through your code modules and completely isolate those that constitute the user interface from those that make up the remainder of your application, then you can focus your migration efforts on the process of replacing the user interface modules with equivalent ones developed for Motif. Many application developers follow software development methods that require this kind of clean

separation and, in some cases, even formally specify the program boundaries between user interface and application internals.

Alternatively, if your software is more monolithic and has application-specific abilities embedded within functions that also provide the user interface, then you may have to spend extra time separating the two types of functions, thereby complicating your migration. In an extreme case, you must choose between violating the style guide or redesigning part of the program.

The amount of time involved in taking full advantage of the Solaris CDE software significantly depends on how your application is laid out. Applications that are well designed are easier to port and to properly break down for maintenance and readability.

Static Versus Dynamic Layout

As mentioned previously, porting your application to Motif is not an object-for-object swap. Such swapping concentrates on the static aspects of your application user interface. Complex applications in particular contain many objects that manage the application infrastructure and make it work in dynamic situations. (Dynamic aspects of your application include, but are not limited to, resizing windows, localization, and font changes.) These *manager widgets* that handle the dynamic geometry in your application are different in the OPEN LOOK and Motif toolkits. Your application will not display the dynamism you want if you try an object-for-object swap of manager widgets. Introducing an appropriate design to handle the dynamic aspects of your application typically increases the complexity of its architecture.

XView does not provide much variety for dynamic layout. It primarily fixes objects at (x, y) positions, which can cause difficulties if an application font is changed or the application is localized. Motif and OLIT provide a variety of geometry manager widgets; however, they have significant differences.

TABLE 6–1 OLIT and Motif Geometry Manager Widgets

OLIT	Motif	Comments
`BulletinBoard`	`XmBulletinBoard`	Basically equivalent; provide static x,y pixel-based positioning for children
`Caption`	(none)	OLIT `Caption` widget provides a way to automatically place a label on one of the four sides of a control widget. To get this functionality in Motif, create a separate `XmRowColumn` widget that contains two children: the label and the control

TABLE 6–1 OLIT and Motif Geometry Manager Widgets *(continued)*

OLIT	Motif	Comments
`ControlArea`	`XmRowColumn`	Both widgets provide a way to lay out children in rows and columns. The OLIT `ControlArea` widget supports aligning `Caption` widget children vertically by colon, which the Motif `XmRowColumn` widget does not.[1] The Motif `XmRowColumn` widget enforces certain size policies on children (usually forces children in a particular row or column to be the same size), but the OLIT `ControlArea` widget does not.
`FooterPanel`	(none)	OLIT `FooterPanel` widget provides a way to lay out a window with a floating footer child at the bottom. The Motif `XmMainWindow` widget can be configured for this layout by setting the "message area child" to be a widget that can be used as a footer (such as an `XmForm` or `XmRowColumn` widget).
`Form`	`XmForm`	Both widgets provide a means for attaching their children relative to each other and relative to the `Form` itself, however each takes a different view of these "attachments." The OLIT `Form` widget provides attachment resources for the x and y dimensions, while the Motif `XmForm` widget provides separate attachments for all four sides (top, bottom, left, right). You can convert OLIT `Form` widget attachments to equivalent `XmForm` widget attachments if both attachment paradigms are well understood. The Motif `XmForm` widget also provides a special type of attachment called "Position." Children can be attached to dynamic positions in the `Form` widget which change as the `Form` widget's size changes. This enables children to be configured to always occupy a certain percentage of the `Form` widget's real estate.

TABLE 6–1 OLIT and Motif Geometry Manager Widgets *(continued)*

OLIT	Motif	Comments
`RubberTile`	(none)	OLIT `RubberTile` widget enables children to be configured to take up a percentage of the `RubberTile` widget's height (if vertically oriented) or width (if horizontally oriented). In Motif you can achieve similar functionality by using the Position-based attachment resources in the `XmForm` widget.
`ScrolledWindow`	`XmScrolledWindow`	Both widgets provide a way to encompass a child widget into a scrollable view port.
(none)	`XmMainWindow`	Motif `XmMainWindow` widget provides a manager that lays out its children into specific areas of the window. These areas include the menu bar area, command area, work area, and message area. OLIT has no equivalent widget.
(none)	`XmPanedWindow`	Motif `XmPanedWindow` widget provides a way to lay out its children in vertically oriented panes. Each of the panes has a sash that can vertically resize the pane. OLIT has no equivalent widget.

1. CDE App Builder provides a geometry manager abstraction called a *group* that enables widgets to be automatically positioned in common layouts, including being vertically aligned by colon. App Builder generates all the code to implement this functionality.

Study the Target Environment

The CDE desktop is quite different from the OpenWindows desktop, although it contains many of the same types of tools. The most obvious difference is that it is a Motif desktop. Read the "Architectural Overview" chapter of *Common Desktop Environment: Programmer's Overview* for a discussion of the end user and development environment architectural structure. Also read about:

- Motif: See "Motif 2.1 Documentation" on page 84 and "Motif Programming" on page 86 for a listing of Motif documentation.

- CDE Look and Feel: Refer to *Common Desktop Environment: Style Guide and Certification Checklist* for details. To be style-guide compliant, your application must pass the checklist at the end of the book.
- CDE Run-time and Development Environments: See “CDE Documentation” on page 83 for a listing of all CDE documentation that Sun provides. In addition, the desktop applications each have online help volumes.

GUI Development Tools

If you used a Motif GUI builder to create the GUI for your application, your migration to the Solaris CDE desktop will probably be easier. In most cases, the use of a builder implies that you have some degree of separation between user interface functions and application internals, the advantages of which were previously discussed.

Also, builders typically use generalized internal storage formats or are capable of generating interchange files, each of which may be post-processed to automate some of the conversion process. Contact your builder vendor to see what migration tools they are currently offering.

Other less tangible tools that you might have used and that could ease your transition include development approaches that produced functional requirements documents or high-level designs. These representations may describe your application in terms less specific to the OPEN LOOK user interface that are more amenable to being mapped to CDE and Solaris Motif than your source code.

Use Transition Tools

No one tool on the market will do everything you need for transitioning your application to the Solaris CDE desktop. Nonetheless, depending on different aspects of your application, you may find tools to solve some of your problems. These specialized tools can make the transition easier.

As you research the tools available to you, ask yourself if any are a good match for your application transition and what you are trying to do.

Tools to Transition to Motif

Many third-party tools are offered to transition to Motif. For example, Integrated Computer Solutions (ICS) offers two converters: one translates XView source code to GIL files and one translates GIL files to Motif source code.

Sun's Devguide offers Motif Conversion Utilities as a way of helping customers make the transition. Devguide's front end is still in the OPEN LOOK user interface, and it has the same OPEN LOOK palette, but you can use the Motif Conversion Utilities to transform GIL files into UIL files or Motif and C code. Use the Motif Conversion Utilities (`guil` and `gmf`) in the Solaris 2.4 Software Developer's Kit to convert GIL files to UIL and Motif C code. UIL files can be imported into Motif GUI builders to generate Motif-based GUIs.

Sun also provides a Motif GUI builder called SunSoft Visual WorkShop.

Note that using transition tools for GUI migration results in a GUI that you will probably need to refine to be CDE style-guide compliant. You may need less time to re-layout the GUI if you use a GUI builder tool.

Tools to Transition to the Solaris CDE Desktop

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. BIL file format is similar to GIL file format. However, BIL files contain CDE-specific information and produce a Solaris Motif GUI. The GIL-to-BIL converter makes some educated conversion guesses based on heuristics, which might not give you the result you want. For best results, take the BIL files that the converter produces, load them into App Builder, and modify the user interface as needed.

App Builder is a tool in the Solaris CDE environment that is very similar to Devguide, which is used in the OpenWindows environment. Transitioning to CDE in this manner solves a significant number of conversion problems. Because App Builder generates files for you, these files are easier to manipulate than C code you would have to write yourself. App Builder preserves application stubs files, so your application-specific code is unchanged. It also uses the same file-naming conventions as Devguide.

If you did not use Devguide, there are currently no third-party tools to use for converting OPEN LOOK applications to CDE. Try a Motif conversation tool such as is discussed in "Tools to Transition to Motif" on page 57 above.

Use Existing Code

This section discusses the ways in which the Solaris CDE software provides code for you to either use or learn from to develop Solaris CDE applications.

GUI Builder Code

Use a Motif GUI builder such as App Builder or SunSoft Visual WorkShop to build a simple Solaris CDE application. Then generate code from it to see what the code looks like. This is a good way to learn how to use Solaris Motif and some of the CDE functionality.

The more features in App Builder you use, the more coverage you get in generated code of Solaris CDE features. Try App Builder to, for example, alter object attributes, create connections, and use the Applications Framework Editor. Then generate code and examine it. Cycle through this process to generate variations of code to look at and learn from.

Demo Code

The Solaris CDE development environment contains demo source code that can considerably ease your application porting tasks.

Each development environment component has a demo directory in `/usr/dt/examples`. The demo directory contains an example program that uses the component's APIs. Read the demo code to learn how to incorporate the component behavior into your application. In some cases you can copy and paste code from the demo into your application.

The `/usr/dt/examples/template` directory contains a demo program that integrates most of the Solaris CDE components that comprise basic and recommended integration functionality. This template demo illustrates how to write a simple application that is well integrated with the Solaris CDE desktop.

Tips

Here are some tips about writing applications for the CDE desktop.

Floating Menus

To ensure that your application works properly on the CDE desktop, any functionality you put into floating menus should also be provided by some pulldown menu. This will enable your application to work with both two-button and three-button mouse devices.

Colormap Behavior

Colormap installation is handled differently under CDE than it was in the OpenWindows environment. This difference is most visible for applications that specify a list of subwindows that use colormaps other than the default colormap. This list of subwindows is specified in the `WM_COLORMAP_WINDOWS` property.

In the OpenWindows environment, applications need only to specify a list of subwindows in this property. As the user moves the pointer around the screen, the OpenWindows window manager (`olwm`) installs the appropriate colormap for whatever window is under the pointer.

The CDE window manager, dtwm, does not provide this behavior. Applications that relied on `olwm`'s pointer-based colormap installation will likely not display in their proper colors when run under CDE. There are several things that you can do to avoid this problem:

- Avoid using the `WM_COLORMAP_WINDOWS` property entirely, and update the colormap attribute of the top-level window (the window of the shell widget) with whatever colormap is appropriate. This colormap will be installed whenever the window is given the input focus.
- The `WM_COLORMAP_WINDOWS` property is an ordered list of windows, and typically only the first window in this list will have its colormap installed and will appear in its proper colors. The other subwindows will most likely appear with incorrect colors. If it is important for a particular subwindow to appear with its correct colors, it should be placed first in the list.
- If the ID of the top-level window does not appear in the list, it is implicitly assumed to appear first. To get a subwindow to display with its correct colors, the ID of the top-level window should appear in the list somewhere after the ID of the subwindow.
- Applications can update the `WM_COLORMAP_WINDOWS` property at any time. If the application's state changes such that a different subwindow should now have its colormap installed, the application should update the property so that the new subwindow appears first.

The user can also modify the key bindings of `dtwm` to bind the `f.next_cmap( )` and `f.prev_cmap( )` functions to keyboard keys. These functions will step forward and backward through the `WM_COLORMAP_WINDOWS` list and install a different colormap each time the user presses the appropriate keys.

Summary: Things to Keep in Mind

Here is a list of things to keep in mind as you think about porting your application to the Solaris CDE desktop:

- OPEN LOOK to Motif migration is a complex and wide-ranging issue

It generally does not amount to an object-for-object swap. Do you really need to port? If so, think of it as an art and not as a science.

- What is your schedule?

  When must you ship? What resources do you have available to you? Can you tie the port in with the next major release?

- How did you build your application?

  Did you use a builder? Were any other tools involved in producing your application? If so, go to your vendor and ask for CDE support.

- Is your application GUI separate from internals?

  Did you use C++ objects to encapsulate the GUI? If so it will be easier to port your application.

- Are you starting from XView or OLIT?

  OLIT applications are generally easier to port because they rely on the Xt intrinsics, as does Motif.

- Are there any customer issues?

  Do you have to be concerned about interoperability with other systems? Are there transition and training issues for the customer? Is a mixed desktop acceptable for now?

CHAPTER 7

Porting Example: OPEN LOOK to Solaris Motif

This chapter presents an example of porting an OPEN LOOK application with a simple graphical user interface (GUI) to Solaris Motif. It illustrates the two-step process of porting: first translating the OPEN LOOK user interface objects one-by-one to Motif objects, then cleaning up the interface and making sure it conforms to the CDE style guide.

Of course, your goal in porting from the OPEN LOOK user interface to Solaris Motif should be to convert the GUI and clean up at the same time. The more familiar you become with the CDE style guide, the easier this will be. Also, as you learn more about Motif, learning the correlation between OPEN LOOK objects and Motif widgets will become easier.

When porting your application, take the opportunity to examine its user interface. Think about whether it could be streamlined or made more user-friendly. In going from the OPEN LOOK user interface to Motif, the application's look will change anyway. This is a good time to re-examine design decisions you made in the past.

OpenWindows 3.4 Snapshot Application

Figure 7–1 is an illustration of the OpenWindows 3.4 Snapshot application. It is running on the Solaris CDE desktop using the Motif Window Manager, so it has a Motif title bar.

The Snapshot GUI is fairly simple. It contains:

- Load..., Save..., Snap, and View... buttons
- Print menu button
- Snap Type and Snap Delay exclusive settings
- Beep and Hide check box settings
- Drop target

Figure 7–1 OpenWindows 3.4 Snapshot Application GUI

Convert

Using App Builder, the Snapshot GUI translates object-for-object into the GUI shown in Figure 7–2:

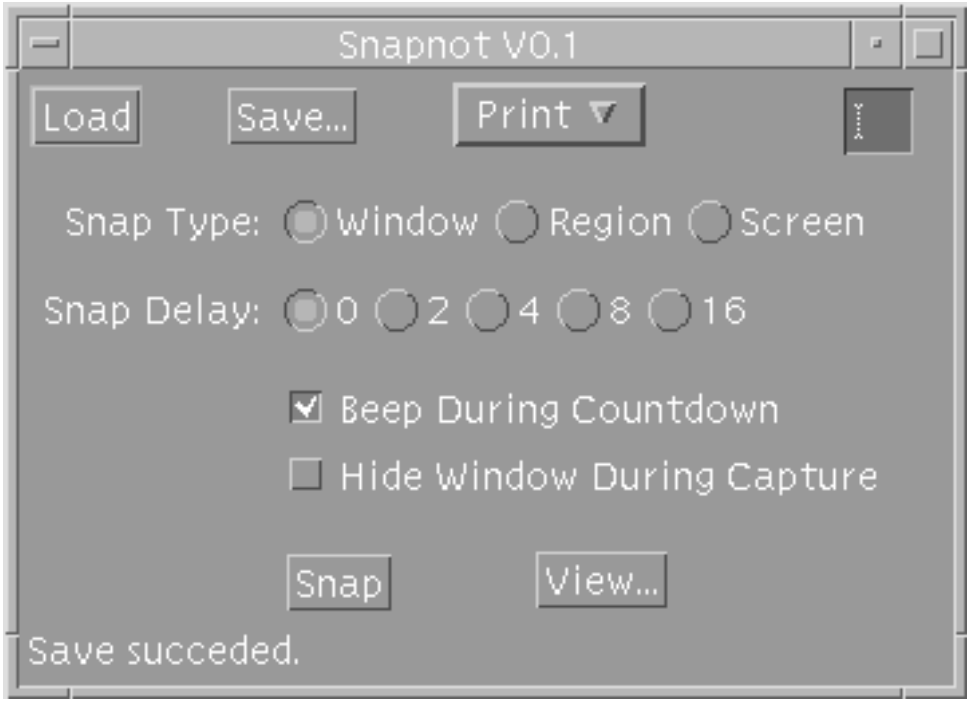


Figure 7–2 Motif Port of Snapshot GUI (Direct Translation)

You can see that the pieces of the interface and the general layout are very similar to the OPEN LOOK version. The buttons remain, except that they are square now instead of round. The check boxes look the same. The OPEN LOOK exclusive settings have changed to Motif radio buttons (which also offer mutually exclusive choices). A small text field with no label replaces the drop target.

Clean Up

If you want, you can consider yourself finished with this application port. No functionality has been lost, and the GUI now appears in Motif. However, you should ask yourself two questions:

- Is the resulting application style-guide compliant?
- Can the GUI be redesigned to make the application more user friendly?

Consider CDE Style Guidelines

Common Desktop Environment: Style Guide and Certification Checklist highly recommends placing menu bars in applications. In addition, it considers File, Options, and Help menus as standard, to be placed in order from left to right. (Other menus are also considered standard but do not apply to the Snapshot application.)

The cleaned-up version places the Load, Save, and Print buttons under the File menu. These are typical file operations. “Beep During Countdown” and “Hide Window During Capture” are options that users have available to them, and are therefore put under the Options menu. This design decision also makes the interface less cluttered, since the text for these check boxes takes up a lot of screen space. A Help menu is added to comply with the style guidelines.

Common Desktop Environment: Style Guide and Certification Checklist calls for a footer message at the bottom of an application whenever status is displayed there. This is added to the Snapshot GUI.

Motif does not support a drop site object. CDE supports a drag source object, which appears in the final GUI as a labelled camera icon. If you want drop support, the CDE style guidelines indicate that you must provide code to enable objects to be dropped anywhere in the application’s main window.

Finally, the Snap button at the bottom is highlighted, indicating that it is the default value (between Snap and View) if the user presses Return instead of pressing a button.

Other Design Considerations

The radio buttons take up a lot of screen space in the Snapshot GUI. CDE provides a widget called a `DtComboBox` that takes up less screen space and provides the same functionality. The Snap Type and Snap Delay radio buttons are replaced by `DtComboBox` widgets. The most typically used values appear as `DtComboBox` defaults.

You cannot use the `DtComboBox` widget if you are developing your application for the OpenWindows desktop. This widget is part of the CDE development environment, not the OpenWindows development environment. Even if you do have access to the ComboBox, however, you may want to retain the radio boxes, depending on issues such as ergonomics, customer needs, and consistency.

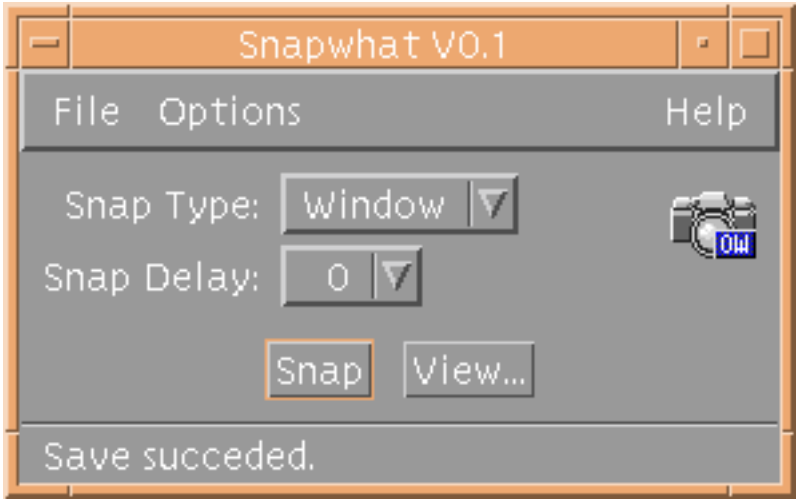


Figure 7–3 Motif Port of Snapshot GUI (Final Port)

APPENDIX **A**

User Interaction Changes

This appendix presents all the user interactions that have changed from the OPEN LOOK style to the CDE style. If a user interaction is not mentioned, the OPEN LOOK and CDE styles are the same.

Use this information to give your CDE application the correct user model.

Note - In this appendix, *CDE Style Guide* refers to *Common Desktop Environment: Style Guide and Certification Checklist*.

TABLE A–1 User Interaction Changes

Category	Topic	OPEN LOOK:	CDE:
Desktop	Multiple desktops	One main workspace or virtual, multiple screen support (OLVWM).	Multiple workspaces are accessible from the Front Panel; users can add, remove, and navigate between workspaces.
	Action feedback	No indicator to provide feedback when application is launched.	Indicator light on Front Panel, which initially blinks when application is first launched and when other actions occur.
	Application access	Applications (and utilities) are accessible from Programs menu off Workspace menu. Programs menu customizable via GUI.	Applications accessible from the Front Panel, the Application Manager, the Personal Applications subpanel, as well as the Programs submenu of the Workspace menu. No GUI for customizing Programs menu.

TABLE A–1 User Interaction Changes *(continued)*

Category	Topic	OPEN LOOK:	CDE:
	Drag and drop to desktop	When icon from File Manager dropped on desktop, the application associated with the icon is launched or executed.	Dropping icon from File Manager onto desktop does not launch or execute: it creates a reference to the icon on the desktop (the original icon still appears in the File Manager).
	Fast help information	Uses Magnify Help; has link to More Help window for further information.	Uses On Item Help to go to item in Help handbook.
	Application Help	Available by pressing keyboard Help key when in application.	Help available from most menu bars (located at the far right of the menu bar) and dialog boxes; also available by pressing the keyboard Help key when in application.
**********	**********	**********	**********
Properties	Multiple terms used to indicate properties	Uses the term Properties (and sometimes, Settings), generally found under Edit menu.	Uses the terms Options and Properties. Options refers to application-specific characteristics and is generally found under the File or Options menu. Properties is used to set object-specific characteristics and is generally found under the Edit menu.
	Global desktop options	Set from the Workspace menu.	Set from the Style Manager application.
**********	**********	**********	**********
Windows	Window titles	Generally follows the form: "File Manager V3.4: /*home*/*username*/ Project_Folder".	Generally follows the form: "File Manager - Project_Folder". Version numbers not included in title (version numbers can be included in the application's About box). Title format specified in *CDE Style Guide* (see pages 92-93).
	Move a window	Click anywhere on window frame (except window corners) and drag to move the window.	Click window title area and drag to move the window.

TABLE A–1 User Interaction Changes *(continued)*

Category	Topic	OPEN LOOK:	CDE:
	Expand a window	Double-click the frame to expand window: window expands vertically. The window position is retained.	Click the maximize button to expand the entire window. The default is maximize equals full screen. To change the maximize direction to vertical only, use the `Dtwm*maximumClientSize:vertical` resource.
	Resize a window	Click any corner of the window and drag to resize.	Click any part of the window frame (including the corners) and drag to resize the window. A resize outline box will display and update the window's size during the resize operation.
	Raise a window	Click either the title bar or the window frame.	Click anywhere in window.
	Remove or dismiss window	Click pushpin to remove (dismiss) windows.	Choose Close from the Window menu or press Alt + F4. OPEN LOOK applications run in CDE do not have pushpins, and are dismissed by the above actions.
	Closing a window to an icon (iconify)	Choose the Close command from the Window menu to iconify. [Window menu is the boxed arrow in upper left corner of window.] Since Close is the default choice in the Window menu, a single-click on the Window menu icon will close the window (if default action in Properties is set to Left Mouse Press: Selects Default Item).	Choose Minimize from the Window menu or click the minimize button at the upper right corner of the window frame.
	Quitting (terminating) an application	Choose the Quit (or Exit) command in Window menu.	Choose Close or Exit command in either the Window or File menu or double-click the Window menu icon to exit application.
**********	**********	**********	**********
Menus	Menu layout type	Uses menu buttons.	Uses menu bar.

TABLE A–1 User Interaction Changes *(continued)*

Category	Topic	OPEN LOOK:	CDE:
	Menu layout (standard)	File-View-Edit-Properties	File-Edit-View-Options-Help
	Keep menu on desktop	Click a pushpin in the menu.	Has tear-off menus.
	Default action from menu	In Workspace Properties, can configure Left Mouse Press to represent Selects Default Item. In this mode, a single click on the menu button chooses the default menu item. The default item is shown in the menu surrounded by a ring.	There is no default menu item.
	Quick menu access	Abbreviated pop-up menu shows available commands. Displayed by clicking mouse button 3 (MENU).	Same.
	Accessing menus from the keyboard	No keyboard mnemonics used to access menus.	Can use keyboard to display and choose from menu. The menu's mnemonic is shown by the underlined character in the menu's name (this may not be the first letter of the menu name). The menu is displayed by holding down Alt + the underlined letter. To access a particular menu option, press the key for the item's mnemonic, or press the down arrow to move to the item and press Return or Spacebar.
**********	**********	**********	**********
Dialog Boxes	File selection (via typing file name or choosing from scrolling list)	Generally offers command windows (for example, New, Load, Load/Save) that enable users to type or select a file.	Offers a standard Common Desktop Environment file selection dialog box that can be modified to accommodate application-specific file navigation/selection options. See *CDE Style Guide* for examples and guidelines.
	Print dialog box	Print dialog boxes vary by application.	Offers a common Print dialog box that can be modified to accommodate application-specific print options. See pages 116-122 of *CDE Style Guide* for examples and guidelines.

TABLE A–1 User Interaction Changes *(continued)*

Category	Topic	OPEN LOOK:	CDE:
	Remove or dismiss dialog box	Property windows are a form of dialog box that use both a pushpin and buttons. The Apply button applies changes and dismisses an unpinned Property window but not a pinned one. To dismiss a pinned window, click the pushpin.	Button actions determine whether dialog box remains up after action taken: Cancel dismisses the dialog box without performing any actions not yet applied, Close dismisses the dialog box, OK applies changes and dismisses the dialog box, and Apply applies changes and does not dismiss the dialog box. No pushpins are used when OPEN LOOK applications are run in CDE. See Chapter 7 of *CDE Style Guide*, "Common Dialogs," for more information.
**********	**********	**********	**********
Controls	Mutually-exclusive choice controls	Uses Exclusive Settings control to present a group of mutually exclusive choices. Appearance: touching rectangles.	Uses radio buttons: see pages 244-245 of *CDE Style Guide* for guidelines. Appearance: round buttons.
	Controls for choosing related items/values	Uses the abbreviated menu button, which displays the current choice without displaying the entire menu. Can either choose a value from the button menu or type a value in a text field and press Return to validate the value.	Uses a ComboBox, a Command Box, or a SpinBox to perform the same functions as the abbreviated menu button types. ComboBox uses a scrollable list box, CommandBox uses a prompt for text input and a list component, and SpinBox presents values as a ring of items that wrap. See the "Controls, Groups, and Models" section of *CDE Style Guide* (starting on page 234) for descriptions and guidelines.
	Toolbars	Does not use toolbars.	Uses toolbars.
	Button shape	Rounded buttons.	Square buttons.
**********	**********	**********	**********
Mouse	Default mouse button functions	Mouse button 1 = Select Mouse button 2 = Adjust Mouse button 3 = Menu	Mouse button 1 = Select Mouse button 2 = Transfer (can be changed to Adjust through Style Manager) Mouse button 3 = Menu

TABLE A–1 User Interaction Changes *(continued)*

Category	Topic	OPEN LOOK:	CDE:
	Extend selection	On a 3-button mouse, use mouse button 2 (also called the ADJUST button) to do an extended selection. Click mouse button 1 at the beginning of the text that you want to select, move pointer to end of desired text, and click mouse button 2; text is highlighted.	To do an extend selection, use Shift key + mouse button 1, or change the behavior of mouse button 2 from Transfer (the default setting) to Adjust through Style Manager.
	Displaying menus	Click mouse button 3 on a menu button to display the menu.	Click a menu bar entry with either mouse button 1 or mouse button 3 to display the menu.
	Opening icons: shortcut	Double-click icons to open.	Double-click icons to open. Single- or double-click Front Panel controls, and single-click Style Manager controls, to launch the related application.
	Quick Cut	To move text to insertion point, hold down the Cut key, select text (a line is drawn through selected text), and release the Cut key. When key is released, text moves to the insertion point.	To move text to insertion point, hold down Alt + Shift + mouse button 2 (Transfer), and select text (a line is drawn under selected text). When combination is released, text moves to the insertion point.
	Quick Paste	To copy to the insertion point, hold down the Paste key, select text (a line is drawn under selected text), and release the Paste key.	To copy to the insertion point, hold down Alt + mouse button 2 (Transfer), and select text (a line is drawn under selected text). When combination is released, text is copied to the insertion point. (This will also work with Alt + Control + mouse button 2.)
**********	**********	**********	**********
Drag and Drop	Drag and drop (direct manipulation)	Uses one set of drag-and-drop methods and visual feedback.	The basics of drag and drop are similar to OPEN LOOK, but the specifics are different. See Chapter 3 of *CDE Style Guide*, "Drag and Drop," for details of drag-and-drop operations, performance guidelines, feedback, and graphics specific to CDE.

TABLE A–1 User Interaction Changes *(continued)*

Category	Topic	OPEN LOOK:	CDE:
	Drop onto minimized icons	Can drop onto minimized icons on desktop; drop will result in appropriate action.	Not supported. Can drop onto Front Panel controls (for example, Printer, Calendar Manager, Trash, Text Editor, Mailer).
	Drag and drop multiple messages from Mail application to File Manager	Creates one mailfile icon per message in File Manager.	Creates one mailbox that contains all the selected messages.
	Drag mail messages to another mailbox window	Not available.	Moves the selected messages to the destination mailbox.
	Drop selected icon into text window	Text appears at drop point.	Text appears at insertion point (same as choosing Include from the File menu).
	Drop zone (target)	Drop zone (target) is a small rectangle typically located in top right of application's control area.	No specific drop zone graphic is used; any editable control can serve as a drop zone. User gets feedback on validity of drop zone.
	Sourcing drags	Uses drag-and-drop source indicator graphic.	Several versions of source indicators are used, representing the type of selection (single or multiple) and the type of object selected. The drag icons are dynamically composed and contain the source indicator information.
**********	**********	**********	**********

TABLE A–1 User Interaction Changes *(continued)*

Category	Topic	OPEN LOOK:	CDE:
Keyboard	Keyboard accelerators and engravings	Uses Meta key in combination with accelerator key. Standard examples: Meta + z = Undo, Meta + c = Copy, Meta + v = Paste, Meta + x = Cut. Uses default set of key bindings and enables customization of key bindings (for example, EMACS).	Uses Control key in combination with accelerator key. Standard examples: Control + z = Undo, Control + c = Copy, Control + v = Paste, Control + x = Cut. Default key bindings sometimes differ from OPEN LOOK key bindings (see Appendix A of *CDE Style Guide* for key engravings). Customization is also available (see Chapter 10 of *CDE User's Guide* for customization information).
	Sun Special keys	Supported in OPEN LOOK.	Supported in Solaris CDE.
	Deleting characters	Can use either Backspace or Delete key.	The Backspace key deletes characters to the left of the cursor; the Delete key deletes characters to the right of the cursor.

APPENDIX **B**

Internationalization and CDE

This is a guide for software developers writing internationalized Solaris CDE applications. It presents the differences between OpenWindows and CDE internationalization. Any steps that must be performed to internationalize a Solaris CDE application that are not mentioned in this appendix remain unchanged from the OpenWindows guidelines. For general information on CDE and internationalization, see *Common Desktop Environment: Internationalization Programmer's Guide*.

Solaris CDE supports all locales that Solaris 2.6 supports.

This appendix assumes familiarity with Xt and Motif programming.

Ensure Correct CDE NLS Environment

You must make sure that the `NLSPATH` environment variable is set properly. This ensures that your application will find message catalogs.

If your application calls `DtInitialize( )`, then `NLSPATH` is set to the CDE default location for finding message catalogs. The default location is `/<CDE_INSTALLATION>/lib/nls/msg/<locale>/<application>.cat`; for example, `/usr/dt/lib/nls/msg/ja/dtcm.cat`.

`DtInitialize( )` sets other variables and performs other tasks. Consult its manpage to determine whether you need this function. `DtInitialize( )` is in `libDtSvc`. If your application does not already link to this library, you may not want to invoke `DtInitialize( )` just to set `NLSPATH`.

If your application does not call `DtInitialize( )`, you must set `NLSPATH` to ensure that your application can find message catalogs.

If your application installs or expects message files to be in a location other than the default location, you should append that location to `NLSPATH`.

Note - Other CDE applications may not be able to find their message catalogs if you reset NLSPATH entirely. It is better to append to NLSPATH if your message file location differs from the default location.

Message Catalog Functions

To be portable to other XPG4 UNIX platforms, Solaris CDE applications must use the `catopen( )`, `catclose( )`, and `catgets( )` family of XPG4 messaging functions (instead of `gettext( )` and so on). See the man pages for information on how to use these functions.

Also, if the system on which you are writing your application does not define `NL_CAT_LOCALE` in a header file, you must define it in your application. This ensures that the application is portable to other UNIX XPG4-compliant platforms.

Locale Announcement

So that your application knows what locale it is running in, use:

- `XtSetLanguageProc( )`—If your application is Motif/Xt GUI-based
- `setlocale( )`—If your application is not GUI-based

Character Strings and XmStrings

This section describes how to convert between character strings and `XmStrings`.

To Convert from Character String to XmString

In Motif, you must make an explicit call to convert a string to `XmString`, an internal representation of the string. If you internationalize your application, use `XmStringCreateLocalized( )` to perform this conversion.

In Solaris Motif—unlike in Motif 1.2.3—`XmStringCreateLocalized( )` recognizes `\n` as a line separator, so a string can have multiple lines.

To Convert from XmString to Character String

To convert from XmString to a character string, you must traverse the compound string and retrieve each segment individually. See section 19.3.3 of the *Motif Programming Manual*, Volume 6A, which contains a code sample that performs this conversion.

Do not use `XmStringGetLtoR( )` to convert from XmString to a character string. This function assumes the text is oriented left-to-right.

Include app-defaults File

If your application uses resources, make sure to include an `app-defaults` file. Refer to the *X Toolkit Intrinsics Programming Manual* (Volume 4, Section 2.3.3) for details on setting up an `app-defaults` file.

Localize Motif Resources

Localize any Motif resource that represents something that must be customized for a particular locale by putting the resource in your `app-defaults` file. These include, but are not limited to:

- XmNmnemonic

- XmNfontList
- XmNcolumns
- XmNinputMethod
- XmNpreeditType

Deliver Message Catalog

Every SVR4 UNIX platform contains the `gencat` utility. Run `gencat` on the translated message catalog file to produce a binary form of the message catalog. The resulting file is a formatted message database.

CDE and gencat

Each platform has its own implementation of `gencat`. Follow these rules for message catalogs to ensure that each message catalog's format does not cause different `gencat` utilities to break:

1. **Use a `$quote` directive if you have trailing spaces (see the `gencat` man page).**
2. **Do not put more than one space between the message id and the message string.**
3. **Insert a space between `$` and the comments.**
4. **Message IDs need to be sorted in ascending order.**

.msg Files

A `.msg` file is a message catalog containing translatable text that appears in your application code. It is the file that you give to a translator for translation. See the `gencat` man page for its format. Use `gencat` to turn a translated message catalog to a message database.

.cat Files

A `.cat` file is a message catalog that results when you run `gencat` on a `.msg` file. This is a binary file.

Fonts

In the OpenWindows environment, font alias names differ depending on the application locale. In Solaris CDE, the font alias names are independent of locale. Refer to *Common Desktop Environment: Internationalization Programmer's Guide* or *Solaris Common Desktop Environment: Programmer's Guide* for information on CDE font names.

Internationalizing Shell Scripts

The `dtdspmsg` utility is useful for internationalizing shell scripts. The `dtdspmsg` command displays a selected message from a message catalog. See the `dtdspmsg` man page for details.

APPENDIX **C**

Recommended Reading

This appendix lists books and articles on issues related to OPEN LOOK, Motif, and Solaris CDE application development. The books are available through Sun Express or through your local bookstore.

CDE Documentation

The Common Desktop Environment documentation set contains documents that will help you in your transition to Solaris CDE. These books will be published soon by Addison-Wesley.

Development Environment

- *Common Desktop Environment: Programmer's Overview*
- *Common Desktop Environment: Style Guide and Certification Checklist*
- *Common Desktop Environment: Application Builder User's Guide*

- *Solaris Common Desktop Environment: Programmer's Guide*
- *Common Desktop Environment: Help System Author's and Programmer's Guide*
- *Common Desktop Environment: ToolTalk Messaging Overview*
- *Common Desktop Environment: Internationalization Programmer's Guide*
- *Common Desktop Environment: Desktop KornShell User's Guide*
- *Common Desktop Environment: Product Glossary*

Online man pages are available for all development environment components.

Run-time Environment

- *Solaris Common Desktop Environment: User's Guide*
- *Solaris Common Desktop Environment: Advanced User's and System Administrator's Guide*
- *Solaris Common Desktop Environment: User's Transition Guide*

Online help volumes are available for most of the run-time environment components.

ToolTalk Documentation

- *The ToolTalk Service: An Inter-Operability Solution*, published by SunSoft Press and PTR Prentice Hall, Englewood Cliffs, NJ 07632, ISBN 0-13-088717-X
- *ToolTalk and Open Protocols: Inter-Application Communication*, by Astrid Julienne and Brian Holtz, published by SunSoft Press and PTR Prentice Hall, Englewood Cliffs, NJ 07632, ISBN 013-031055-7

The Solaris 7 Software Developer AnswerBook set contains:

- *ToolTalk Reference Manual*
- *ToolTalk User's Guide*

Motif 2.1 Documentation

The documentation listed in this section describes Motif 2.1 interfaces. You can purchase these books at your local bookstore.

- OSF Application Environment Specification (AES) User Environment Volume, Revision C, PTR Prentice Hall, 1993.

Describes the Motif Application Environment Specification. Information on these specifications can also be found in the *Motif 2.1 - Programmer's Reference.*

- *CDE 2.1/Motif 2.1 User's Guide* from The Open Group (`www.opengroup.org`)

  This guide describes the basic features of both the Motif user environment and the Common Desktop Environment. It explains how a user interacts with these features and with Motif and CDE-based applications, as well as how to customize Motif environments and the CDE and how to use the Information Manager to access, read, and search online documentation.

- *Motif 2.1 - Programmer's Reference* from The Open Group (`www.opengroup.org`)

  This programmer's reference manual provides reference information for the Motif commands and functions in UNIX-style manual pages. The reference information covers the toolkit (library functions, widget documentation, and resources), the window manager, UI language commands, and the library functions.

- *Motif 2.1 - Programmer's Guide* from The Open Group (`www.opengroup.org`)

  These three volumes provide detailed reference descriptions of all Motif programs, Xt widget classes, Xm widget classes, translations, Xm data types and functions, Mrm functions, Uil functions, and file formats.

- *CDE 2.1/Motif 2.1 - Style Guide and Glossary* from The Open Group (`www.opengroup.org`)

  This guide provides developers who design and implement new products with a framework of behavior specifications that is consistent with the Motif and Common Desktop Environment (CDE) user interface. This behavior is established by drawing out the common elements from a variety of current behavioral models. The document also includes a comprehensive glossary of terms used in the Desktop Documentation set.

Graphical User Interfaces

OPEN LOOK to Motif GUI Transition Guide 801,6567-10, SunSoft, October, 1993.

OPEN LOOK Graphical User Interface: Programmer's Guide, UNIX System Laboratories, 1992.

OPEN LOOK Graphical User Interface: User's Guide, UNIX System Laboratories, 1992.

OPEN LOOK Graphical User Interface Application Style Guidelines, Sun Microsystems, Inc., Addison-Wesley Publishing Company, Inc., 1990.

OPEN LOOK Graphical User Interface Functional Specification, Sun Microsystems, Inc., Addison-Wesley Publishing Company, Inc., 1990.

OPEN LOOK Graphical User Interface: Programmer's Reference Manual, Prentice Hall, 1992.

OPEN LOOK Intrinsics Toolkit Widget Set Programmer's Guide, AT&T, 1990.

OPEN LOOK Intrinsics Toolkit Widget Set Reference Manual, AT&T, 1990.

OSF/Motif Programmer's Guide, Revision 1.2, Open Software Foundation, Prentice Hall, 1993.

OSF/Motif Programmer's Reference, Revision 1.2, Open Software Foundation, Prentice Hall, 1993.

OSF/Motif Style Guide, Revision 1.2, Open Software Foundation, Prentice Hall, 1993.

OLIT & Motif: A Technical Comparison, Amy Moore, M. Goyal, SunSoft, September, 1992.

Motif Programming

Motif Programming in the X Window System Environment, William A. Parrette, McGraw-Hill, 1993.

Motif Programming: The Essentials—and More, Marshall Brain, Digital Press, 1992.

Motif Programming Manual, Dan Heller, O'Reilly & Associates, 1992.

Motif Reference Manual, Paula Ferguson, O'Reilly & Associates, 1992.

The X Window System Programming and Applications with Xt, OSF / Motif Edition, Douglas Young, Prentice Hall, 1990.

OPEN LOOK Programming

An OPEN LOOK at Unix: A Developer's Guide to X. John David Miller, M&T Books, 1990.

The X Window System Programming and Applications with Xt, OPEN LOOK Edition, Douglas Young and John A. Pew, Prentice Hall, 1992.

XView Programming Manual, Dan Heller, O'Reilly & Associates, Inc., 1991.

XView Reference Manual, Thomas Van Raalte (ed.), O'Reilly & Associates, Inc., 1991.

Xt/XLib Programming

Programmer's Supplement for Release 5 of the X Window System, Version 11, David Flanagan, O'Reilly & Associates, Inc., 1991.

X Toolkit Intrinsics Programming Manual, Adrian Nye and Tim O'Reilly, O'Reilly & Associates, Inc., 1990.

X Window System Toolkit, Paul J. Asente and Ralph R. Swick, Digital Press, 1990.

X Window System, X Version 11 Release 5, Third Edition, Digital Press, 1992.

X Toolkit Intrinsics Reference Manual, O'Reilly & Associates, Inc., 1991.

Xlib: C Language X Interface, James Gettys, Robert W. Scheifler, Ron Newman, Silicon Press, 1989.

Xlib Programming Manual, Adrian Nye, O'Reilly & Associates, Inc., 1990.

Xlib Reference Manual, Adrian Nye (ed.), O'Reilly & Associates, Inc., 1990.

Index

X