Porting Issues and Ideas 6

This chapter discusses porting OPEN LOOK applications to CDE 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	*46*
Do You Need to Port?	*46*
If You Decide to Port	*47*
Start Small	*49*
Architectural Impact	*50*
Use Transition Tools	*53*
Use Existing Code	*55*
Tips	*56*
Summary: Things to Keep in Mind	*57*

6

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, "Development Environment Transition Issues," for a summary of some of these features. See *CDE 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 5, 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" and "Optional Integration" below) if you want to use them and are willing to modify your code.

See *CDE 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 *CDE 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:

- CDE 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 Application Builder" 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, "Porting Example: OPEN LOOK to CDE Motif," takes a simple OPEN LOOK application and illustrates this process.

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

Use Application Builder

Use Application Builder (App Builder) 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." 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 App 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 App Builder. App Builder produces all the GUI and application framework code, which frees your time to focus on application code.

6

Architectural Impact

Resist the urge to dive right in and start transforming your code to CDE 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 CDE 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
`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).

Table 6-1 OLIT and Motif Geometry Manager Widgets

OLIT	Motif	Comments
`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.
`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 *CDE Programmer's Overview* for a discussion of the end user and development

environment architectural structure. Also read about:

- Motif: See"Motif 1.2.3 Documentation" on page 77 and "Motif Programming" on page 79 for a listing of Motif documentation.
- CDE Look and Feel: Refer to *CDE 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 76 for a listing of all CDE documentation that SunSoft provides. In addition, the desktop applications each have online help volumes.

GUI Development Tools

If you used an application 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 CDE 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.

6

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.

SunSoft'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.

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 CDE 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" 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.

Application Builder

Use App Builder 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 CDE 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.

6

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?